ACKNOWLEDGEMENTS

Many people have contributed to the quality of this thesis. First of all, my supervisors, *Antoon Kolen* and *Alexander Rinnooy Kan*, have been especially helpful. Their different personalities gave me the opportunity to benefit from two different viewpoints of our field during the last five years. Alexander was the one who always kept a good overview of my research, and he showed me directions that I had never thought of. Moreover, in his few moments of spare time, he kept on looking for the right people to talk to, and for the right trips to make. Antoon was the one who wanted to be informed on the details of my work. Reluctant in his belief of correctness, he sorted out all the pieces of information to give advice on how to improve the formulations and proofs of many theorems. Due to him, the final version of this thesis completely outperforms earlier versions. The patience that both supervisors had with me is beyond imagination.

The other members of my Ph.D. committee also contributed more than one may expect. *Laurence Wolsey* was the one who spent a lot of his precious time, patiently explaining the subtleties of the theory of polyhedral combinatorics, and *Luk Van Wassenhove* raised my interest in discrete lot sizing and scheduling problems. This turned out to be a fruitful research area. He also provided me with a lot of valuable information on the subject.

Then, of course, there is *Albert Wagelmans*. He was a valuable and reliable member of the *Dynamic Duo*, both in research and in daily life. We shared many first experiences, and it was just by bad luck that he could not be present at my wedding, but he is well forgiven. The major part of this thesis consists of joint work with this best buddy.

Most of the time has been spent at the Econometric Institute of Erasmus University, with many enthusiastic colleagues. *Auke Woerlee* is the typical example of a rough diamond. In the presence of his colleagues, he could not resist showing that "subtle" remarks were his main talent. However, he was always available to provide help, when I needed it. After the discovery of the

other sexe, he also showed a well-developed sensitivity in his character. *Olaf Flippo* was a master in doing things in his way. He was always generous in sharing his resources with other people. *Cees Dert* was the organizer of many social events. *Guus Boender* often revealed to me that he is an expert in sharing secrets. Also, special attention should be given to *Vitaly Strusevitch*, whom I got to know as someone with great wisdom and many skills. Fortunately, he can enjoy his talents himself. Other colleagues that made my presence more than worthwile are *Ana Isabel de Barros*, *Joaquin Gromicho*, *Gerard Kindervater*, *Harrie Trienekens*, *Andre van Vliet*, *Mario van Vliet*, and *Shuzong Zhang*.

After two years, I got acquainted to people of another institute at Erasmus University, the business department. *Marc Salomon* is one of the researchers of that institute with a good understanding of what is important and what not. I appreciated working with him and his colleagues *Roelof Kuik* and *Leo Kroon*. Especially, the dinner parties led to a major breakthrough in the art of microwave cooking.

During my trip to the Bonn institute of Econometrics and OR, I found out how valuable contacts with other researchers are. I met a lot of very good researchers, from whom I learned a lot. At CORE, in Louvain-la-Neuve, I also met some nice people. I will never forget *Karen Aardal*, whose life seems to be a continuous movie of unexpected events.

The colleagues of Tilburg University and Eindhoven University of Technology are thanked for their patience with me when I was working on my thesis instead of doing my job as a teacher. *Jaap Geerdink* from Eindhoven is thanked for the design of the picture on the front page.

Also many people have contributed to the quality of my life during the last five years. This is especially true for my wife *Carin*, who entered into my life with great enthusiasm to never disappear. Her understanding and her way of handling my thesis problems helped me through the past months. Unfortunately, there were only two colleagues present at our wedding. Therefore, this is perhaps the best place to thank them, *Edwin Romeijn* and *Marc Goedhart*, for their duties as best men.

The friends that I have known for a long time now continue to be a rich source of joy and understanding. This is the reason that two of them have an official role in my defence, *Peter Brock* and my younger brother *Perry*. I know that questions that cannot be answered by me are in good hands with them. This is also the right place to thank *Peter Zwaneveld*, who did not hesitate a second to let me test his computer when I asked him.

Finally, I should thank my *parents*, who were the most important people in my entire life. They were reliable sources of faith and confidence, and they were generous sponsors of my non-intellectual activities. Without them, I would not have had the opportunities to become what I am now.

This research was supported by the Netherlands foundation for scientific research (NWO) from september 1988 until februari 1990, and by the ERASMUS exchange program from september 1988 until april 1989.

CONTENTS

Chapter 1

MODELS AND SOLUTION TECHNIQUES FOR LOT SIZING PROBLEMS

The problems that occur in operations research typically originate in practice, in organizations where profits should be maximized and the availability of resources is limited. In this manuscript, the class of production problems is described, where one or more items are produced on a machine. The main decision problem in this class is the determination of the number of units of a certain item to be produced in a number of periods, the so-called lot sizes. As in all optimization problems, there are factors between which the right balance should be found. In the case of lot sizing problems, this balance should be found between the different types of costs. There is always a fixed cost component, which may consist of costs of manpower and materials for the preparation of the machine in order to start it up. This fixed cost component suggests large lot sizes, in order to reduce the fixed costs per unit of the item. The variable costs are dependent on the number of units to be produced: there are the unit production costs, which may consist of the costs for raw materials and the energy necessary for production, and the unit holding costs for keeping the item units in stock. The variable cost component suggests small lot sizes.

In section 1.1, two basic lot sizing problems are introduced. Here, the different types of fixed and variable costs will be discussed. An important tool to tackle problems in Operations Research is the modelling of a problem. In section 1.2, several integer linear programming models for both problems and some extensions, will be presented. In section 1.3, the standard dynamic programming formulations will be described, with recursion formulae with which these problems can be solved. Section 1.4 is of a different nature. Here the basic theory of polyhedral combinatorics is presented. The reason for this is

that a part of this thesis, the last two chapters, consists of results of this nature. Here, linear constraints are added to linear relaxations of the integer linear programming formulations of lot sizing problems. This chapter ends with section 1.5, in which a detailed overview is provided of the contents of all succeeding chapters.

1.1. Two single-item lot sizing problems.

Developing a production scheme, which finds an optimal balance between the fixed costs and the variable costs is a primary issue in all lot sizing problems. A first mathematical contribution to the solution of a lot sizing problem is the famous EOQ (Economic Order Quantity) formula of Harris [18]. (See also Erlenkotter [12] for a nice historical review.) The EOQ-formula solves the lot sizing problem, where demand is constant over time, and production takes place at arbitrary discrete points in time, without capacity limits.

One of the major limitations of the above model is the restriction that the demand pattern is continuous and constant over time. Therefore, a finite time horizon was introduced by Manne [33], and Wagner and Whitin [54]. This horizon is partitioned in a number of periods, each with its own deterministic demand, independent of the demand in other periods. This problem is called the Economic Lot Sizing problem (ELS). Wagner and Whitin developed a Dynamic Programming algorithm for the single-item version of the problem, where no limitation on production in each period is incurred. Manne used the same structural property as Wagner and Whitin did, (the so-called Zero-Inventory property), to develop a column generation technique to solve the multi-item version of the problem. Moreover, he introduced upper bounds on the production capacities in the periods. In this thesis, however, only the uncapacitated single-item version of ELS is considered. Other major contributions to the economic lot sizing problem are from Zangwill [60] and [61], who studies several extensions of the uncapacitated economic lot sizing problem, and from Lundin and Morton [31], who were among the first to extend the knowledge about structural properties of ELS.

The second problem, called the Discrete Lot Sizing and Scheduling problem (DLS), has two major differences with respect to the Economic Lot Sizing problem. One of these differences is that in each period the decision on the quantity to produce is limited to two choices: either produce at full capacity, or not at all. This is typical in problems where the period lengths are rather short, like days, and the only decision to be made is to produce or not to produce. This type of production policy is called an all-or-nothing policy. Because the periods are usually of the same length the fixed amount of production is equal in all production periods. A consequence of this is that the necessary number of production periods up to a certain period t can easily be deduced from the total demand up to t. Since this can be done for arbitrary periods, demand and production can be normalized to binary parameters and decision variables, respectively. The second difference concerns the fixed cost structure. Since the two different types of fixed costs return in the models in later chapters, in various combinations in both the economic lot sizing problem and the discrete lot sizing and scheduling problem, we discuss this feature in more detail. Schrage [48] was the first who distinguished between costs necessary to start a machine running in a period (the set-up costs) and the costs to prepare the machine to be able to produce the item (the start-up costs). With regard to the start-up costs, one might think of tools to be put on the machine to perform some operations. Therefore, the start-up costs are incurred only at the beginning of a set of consecutive production periods, contrary to the set-up costs which are incurred in each production period. The start-up costs typically arise in case multiple items are produced, and therefore they are also known as change-over costs. Karmarkar and Schrage [23], Karmarkar, Kekre and Kekre [24], and van Wassenhove and Vanderhenst [55] describe algorithms for DLS, and its multiple-item extension. We will consider the following variants of the single-item versions of ELS and DLS, with regard to the fixed costs.

1) ELS with set-up costs (the basic model);
2) ELS with set-up costs and start-up costs;
3) DLS with start-up costs (the basic model);
4) DLS with set-up costs and start-up costs.

In case set-up and start-up costs are combined, then a start-up is only necessary in the case that there is a set-up in the current period, and no set-up in the previous period.

Both problems, ELS and DLS, can be generalized in another way, i.e., by allowing backlogged demand at a certain cost. However, these backlogging models only get limited attention in this thesis. Actually, the only real contribution to the solution techniques for these problems concerns an improved dynamic programming algorithm for the backlogging extension of ELS. Nevertheless, the modelling of these problems is also discussed in the following sections.

Except for the given generalizations of ELS and DLS, there are many more. Most of these, however, make the problem much more difficult to solve. We give some additional structural properties of the most important generalizations below.

1) Capacity constraints on production and inventory;
2) Different items to be produced on the same machine;
3) Different machines that can or must be used to produce an item.

All these generalizations contain ELS or DLS as a subproblem. Therefore, models and solution techniques for ELS or DLS are of great importance for these more general problems, which are NP-hard in general. This has been shown in Florian et al. [15] and in Bitran and Yanasse [7].

1.2. Models for ELS and DLS and extensions.

A straightforward way to model ELS is by introducing variables for the set-ups in each period, variables for the production in each period, and variables for the inventory at the end of each period. With each type of variables a cost parameter is attached. The symbols that are used for the model of ELS are introduced below.

The length of the planning horizon, the number of periods involved, is denoted by T. The set-up costs are denoted by f_t $(t=1,\ldots,T)$, and they are incurred whenever production takes place in period t. This is modelled by a binary variable y_t, which has value one if production takes place, and value zero otherwise. The production costs are denoted by p_t $(t=1,\ldots,T)$ and the inventory costs by h_t $(t=1,\ldots,T)$. The corresponding variables for production and inventory are x_t $(t=1,\ldots,T)$ and I_t $(t=1,\ldots,T)$, respectively. The model does not allow any starting or ending inventory.

$$\text{(ELS)} \qquad \min \sum_{t=1}^{T} (f_t y_t + p_t x_t + h_t I_t) \qquad (1.2.1)$$

s.t.

$$x_t + I_{t-1} = d_t + I_t \qquad (1 \le t \le T) \qquad (1.2.2)$$

$$I_0 = 0; I_T = 0 \qquad (1.2.3)$$

$$x_t > 0 \Rightarrow y_t = 1 \qquad (1 \le t \le T) \qquad (1.2.4)$$

$$x_t \ge 0 \qquad (1 \le t \le T) \qquad (1.2.5)$$

$$I_t \ge 0 \qquad (1 \le t \le T-1) \qquad (1.2.6)$$

$$y_t \in \{0,1\} \qquad (1 \le t \le T) \qquad (1.2.7)$$

The constraints 1.2.4 enforce a set-up in any period with positive production. They can be simplified, using knowledge about upper bounds on the production in each period. Suppose that an upper bound M_t is given on the production in period t. Then the inequality $x_t \le M_t y_t$ correctly models the set-up structure of ELS, since zero production does not imply any restrictions on the set-up variable, and positive production forces the set-up variable to take on the value 1, since it is a binary variable. In the above model it is easy to derive an upper bound for the production in period t, since the ending inventory is forced to be zero. Therefore, an upperbound is $\sum_{\tau=t}^{T} d_\tau$, the cumulative demand of the periods starting with t. For notational reasons, we introduce notation for the cumulative demand of a set of consecutive periods. By d_{st} we denote the cumulative demand of the periods $\{s,\ldots,t\}$, i.e., $d_{st} \equiv \sum_{\tau=s}^{t} d_\tau$.

A more important reformulation concerns the balance equations 1.2.2. These equations state that the number of items that is available in a period, the

ending inventory of the previous period plus the production of the current period equals the number of items that are "used" in the period, the demand and the ending inventory. It follows that $I_t = (\sum_{\tau=1}^{t} x_\tau) - d_{1t}$. Thus, the inventory variables can be eliminated from the formulation. It suffices to add the constraints, that force the production up to period t to be enough to satisfy the demand in the first t periods. This leads to the following model for ELS.

$$\text{(ELS)} \qquad \min \sum_{t=1}^{T} (f_t y_t + c_t x_t) - \sum_{t=1}^{T} h_t d_{1t} \qquad (1.2.8)$$

$$\text{s.t.} \qquad \sum_{\tau=1}^{T} x_\tau = d_{1T} \qquad (1.2.9)$$

$$\sum_{\tau=1}^{t} x_\tau \geq d_{1t} \qquad (1 \leq t \leq T) \qquad (1.2.10)$$

$$0 \leq x_t \leq d_{tT} y_t \qquad (1 \leq t \leq T) \qquad (1.2.11)$$

$$y_t \in \{0,1\} \qquad (1 \leq t \leq T) \qquad (1.2.12)$$

Here, $c_t \equiv p_t + h_{tT}$. The costs c_t are the so-called inventory incremented unit production costs. Note that the second part of the objective function is a constant. It will therefore be omitted in future models. The equation 1.2.9 follows from 1.2.3, and the inequalities 1.2.10 follow from 1.2.6.

The linear programming relaxation of the above model is not very accurate. This is due to the constraints 1.2.11. It is well-known that they do not describe the set-up feature of ELS properly, since d_{tT} is usually a very high upper bound; the production in a period rarely equals this amount. A way to get rid of this inaccuracy is to split the production variables x_t into variables $x_{t\tau}$ $(\tau = t, \ldots, T)$, where τ denotes the period for which the production takes place. For these variables a trivial upper bound is the demand in period τ, i.e., d_τ. It leads to the following model, which resembles the model for the uncapacitated facility location problem. Therefore, it is called ELS-UFL.

$$\text{(ELS-UFL)} \qquad \min \sum_{t=1}^{T} \left(f_t y_t + c_t \left(\sum_{\tau=t}^{T} x_{t\tau} \right) \right) \tag{1.2.13}$$

$$\text{s.t.} \qquad \sum_{t=1}^{\tau} x_{t\tau} = d_\tau \qquad (1 \le \tau \le T) \tag{1.2.14}$$

$$0 \le x_{t\tau} \le d_\tau y_t \qquad (1 \le t \le \tau \le T) \tag{1.2.15}$$

$$y_t \in \{0,1\} \qquad (1 \le t \le T) \tag{1.2.16}$$

The relation with the uncapacitated facility location problem is the following. Consider T facilities from which clients can be served. The decision to be made is to open or not to open a certain facility t. The cost for opening a facility is denoted by f_t. The cost of serving a client τ $(\tau \ge t)$ is c_t per unit of the item, i.e., this cost is only dependent on the facility and not on the client. This is where the major difference with the usual facility location problem lies. In fact, this special cost structure causes that the problem becomes polynomially solvable, in contrast to the original problem which is well-known to be NP-hard.

The strength of this formulation is that one can relax the integrality constraints on the set-up variables, without loosing much information. The linear programming relaxation always yields integer solutions. An algorithmic proof of this can be found in chapter 2. The result is due to Krarup and Bilde [27].

We proceed by stating the models in the original variables for the economic lot sizing problem with start-up costs (ELSS), and the economic lot sizing problem with backlogging (ELSB).

The start-up variables in ELSS are denoted by z_t $(t = 1, \ldots, T)$ and the start-up costs are denoted by g_t $(t = 1, \ldots, T)$. Since a start-up is incurred in period t, only when there is a set-up in period t, and no set-up in period $t-1$, the constraints 1.2.21 are added to the formulation of ELS.

$$\text{(ELSS)} \qquad \min \sum_{t=1}^{T} (g_t z_t + f_t y_t + c_t x_t) \tag{1.2.17}$$

$$\text{s.t.} \quad \sum_{\tau=1}^{T} x_\tau = d_{1T} \tag{1.2.18}$$

$$\sum_{\tau=1}^{t} x_\tau \geq d_{1t} \qquad (1 \leq t \leq T) \tag{1.2.19}$$

$$0 \leq x_t \leq d_{tT} y_t \qquad (1 \leq t \leq T) \tag{1.2.20}$$

$$y_t \leq y_{t-1} + z_t \qquad (1 \leq t \leq T) \tag{1.2.21}$$

$$y_t, z_t \in \{0,1\} \qquad (1 \leq t \leq T) \tag{1.2.22}$$

In the model for ELSB some new parameters and variables are introduced to model the backlogging facilities. The backlogged demand is denoted by I_t^-, and the backlogging costs per unit are denoted by h_t^-. The inventory is denoted by I_t^+, and the inventory costs by h_t^+.

$$\text{(ELSB)} \qquad \min \sum_{t=1}^{T} (f_t y_t + p_t x_t + h_t^+ I_t^+ + h_t^- I_t^-) \tag{1.2.23}$$

$$\text{s.t.} \quad x_t + I_{t-1}^+ + I_t^- = d_t + I_t^+ + I_{t-1}^- \qquad (1 \leq t \leq T) \tag{1.2.24}$$

$$I_0^+ = I_0^- = I_T^+ = I_T^- = 0 \tag{1.2.25}$$

$$0 \leq x_t \leq d_{1T} y_t \qquad (1 \leq t \leq T) \tag{1.2.26}$$

$$I_t^+, I_t^- \geq 0 \qquad (1 \leq t \leq T) \tag{1.2.27}$$

$$y_t \in \{0,1\} \qquad (1 \leq t \leq T) \tag{1.2.28}$$

The constraints are very similar to those of ELS. The two major differences are the new upper bound on production, which is d_{1T} for each period, and the backlogging variables are added to the balance equations 1.2.24. Both ELSS and ELSB can also be modelled as uncapacitated plant location problems. The model for ELSS will appear in chapter 6.

An overview of the variables and parameters is provided below.

parameters:

T: the length of the planning horizon;

d_t $(t=1,\ldots,T)$: the demand in period t;

g_t $(t=1,\ldots,T)$: the start-up cost in period t;

f_t $(t=1,\ldots,T)$: the set-up cost in period t;

p_t $(t=1,\ldots,T)$: the unit production cost in period t;

c_t $(t=1,\ldots,T)$: the unit inventory incremented production cost in t;
h_t^+ $(t=1,\ldots,T)$: the unit inventory cost in period t;
h_t^- $(t=1,\ldots,T)$: the unit backlogging cost in period t.

For all these parameters we use the following convention. For any consecutive set of periods $s,\ldots,t$ the sum of the values of a certain parameter α is written in short:

$$\alpha_{st} := \sum_{\tau=s}^{t} \alpha_\tau$$

variables:

z_t $(t=1,\ldots,T)$: $\begin{cases} 1 \text{ if a start-up is incurred in period } t; \\ 0 \text{ otherwise;} \end{cases}$

y_t $(t=1,\ldots,T)$: $\begin{cases} 1 \text{ if a set-up is incurred in period } t; \\ 0 \text{ otherwise;} \end{cases}$

x_t $(t=1,\ldots,T)$: the production in period t;
I_t^+ $(t=0,\ldots,T)$: the inventory level at the end of period t;
I_t^- $(t=0,\ldots,T)$: the backlogging level at the end of period t.

For the variables the above convention is NOT used, to avoid confusion with the production variables of the uncapacitated facility location problem.

This section ends with the integer linear programming formulations of DLS and DLSS. These models use the same variables for the same type of structure, i.e., the set-up variables, the start-up variables, and the production variables, as well as their corresponding cost parameters all bear the same names as they do in ELS and its extensions.

$$\text{(DLS)} \qquad \min \sum_{t=1}^{T} (g_t z_t + p_t x_t) \tag{1.2.29}$$

$$\text{s.t.} \qquad \sum_{\tau=1}^{T} x_\tau = d_{1T} \tag{1.2.30}$$

$$\sum_{\tau=1}^{t} x_\tau \geq d_{1t} \qquad (1 \leq t \leq T) \tag{1.2.31}$$

$$x_t \leq x_{t-1} + z_t \qquad (1 \leq t \leq T) \tag{1.2.32}$$

$$x_t, z_t \in \{0,1\} \qquad (1 \leq t \leq T) \tag{1.2.33}$$

Note that the production variables are binary, as well as the demands. It is not hard to generalize this model for the extensions with set-ups and/or backlogging costs. A formulation in which the production variables are split, is also possible for DLS. This formulation will be provided in chapter 7.

In the formulation of DLSS the role of the x_t-variables in DLS is taken over by the y_t-variables.

$$\text{(DLSS)} \qquad \min \sum_{t=1}^{T} (g_t z_t + f_t y_t + p_t x_t) \tag{1.2.34}$$

$$\text{s.t.} \qquad \sum_{\tau=1}^{T} x_\tau = d_{1T} \tag{1.2.35}$$

$$\sum_{\tau=1}^{t} x_\tau \geq d_{1t} \qquad (1 \leq t \leq T) \tag{1.2.36}$$

$$x_t \leq y_t \qquad (1 \leq t \leq T) \tag{1.2.37}$$

$$y_t \leq y_{t-1} + z_t \qquad (1 \leq t \leq T) \tag{1.2.38}$$

$$x_t, y_t, z_t \in \{0,1\} \qquad (1 \leq t \leq T) \tag{1.2.39}$$

1.3. Dynamic programming formulations for ELS and DLS.

The basic solution methods for ELS and DLS are based on dynamic programming, like the original algorithm of Wagner and Whitin [54] for ELS. Therefore, this technique will be reviewed first. This is done in a quite informal way. Bellman [5] is usually quoted as the founder of dynamic programming in combinatorial optimization. He is regarded as the first who proposed the "principle of optimality" on which the solution technique for dynamic programming is based. For a more detailed description of dynamic programming and its uses, the reader is referred to Denardo [8]. We will restrict ourselves to an informal description of dynamic programs, that is suitable for the lot sizing problems considered here.

A dynamic program corresponds to a set of states, which contain specific characteristics of the problem at hand. Usually these states are "partial solutions" of the problem. These states are represented by vertices in a

network. Two states may or may not be connected by an arc. In our terminology an arc is defined from one state i to another state j, if state i is "contained" in state j. The costs related to an arc connecting i and j are equivalent to the costs for expanding state i to j. They are denoted by c_{ij}. The network is acyclic. Because the network is acyclic the states (say there are n states) can be numbered from 1 to n, such that all arcs are of the form (i,j) where $i<j$, i.e., a state can only be reached from a lower numbered state.

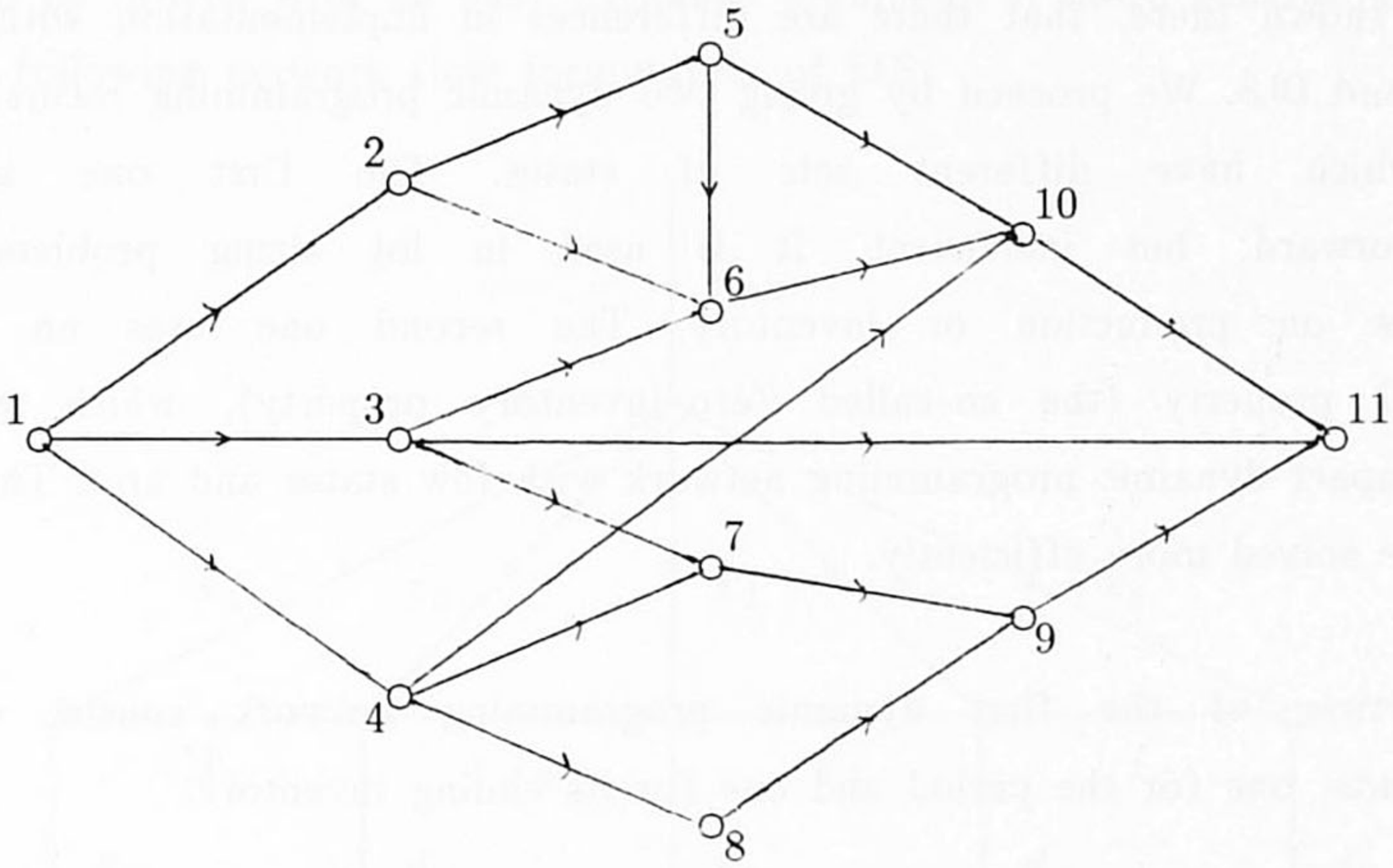


Figure 1.1. *Acyclic dynamic programming network.*

An algorithm to solve the dynamic program, simply finds a shortest path from the initial state to the end state in the network. To facilitate this the costs of reaching a state are defined as follows. The cost of a state F_j $(j=1,\ldots,n)$ is the minimum cost of reaching state j from the initial state, the following recursion can be used to calculate all values F_j $(j=1,\ldots,n)$:

$$F_j := \min_{(i,j)\in A} \{F_i + c_{ij}\}$$

where A is the set of arcs in the network. To calculate all values F_j $(j=1,\ldots,n)$ takes a number of elementary steps of the order of the number of arcs, since each arc is reviewed exactly once. Note that the value F_n gives

the optimal value of the problem. Another way of solving the dynamic program is by calculating the shortest paths in the network in a backward fashion. In that case, a similar recursion formula can be given by defining B_i $(i=1,\ldots,n)$ to be the cost of a shortest path from state i to n, the final state:

$$B_i := \min_{(i,j)\in A} \{c_{ij}+B_j\}$$

Both types of recursions will be used extensively in the chapters 3, 4 and 5. Although there are no conceptual differences between the two recursions, it will be shown there, that there are differences in implementation with regard to ELS and DLS. We proceed by giving two dynamic programming recursions for ELS, which have different sets of states. The first one is very straightforward, but inefficient. It is used in lot sizing problems with capacities on production or inventory. The second one uses an elegant structural property (the so-called Zero-Inventory property), which yields a very compact dynamic programming network with few states and arcs. Therefore, it can be solved more efficiently.

The vertices of the first dynamic programming network consist of two components, one for the period and one for its ending inventory:

$$(t, I_t) \qquad (t=0,\ldots,T; I_t=0,\ldots,D)$$

Here D is the cumulative demand of the periods $1,\ldots,T$, i.e., $D := \sum_{t=1}^{T} d_t$. The initial state is $(0,0)$ and the final state is $(T,0)$. The arcs from state $(t-1, I_{t-1})$ point to the vertices (t, I_t) for $I_t \geq \max\{I_{t-1}-d_t, 0\}$. The costs involved with these arcs depend on whether production takes place or not: the costs are $f_t+p_t x_t$ if $x_t = I_t+d_t-I_{t-1}>0$ and zero if $x_t = I_t+d_t-I_{t-1}=0$. From each state corresponding to period t at most D arcs leave. The number of states of period t is bounded by D as well, which gives the algorithm a worst case running time of $O(TD^2)$, since the complexity of the algorithm is bounded by the number of arcs in the dynamic programming network.

We can do much better by using a structural property of the economic lot sizing problem, the so-called Zero-Inventory property. This property

characterizes optimal schedules in ELS, and it is therefore suitable for the design of efficient algorithms. In the case of ELS itself it even yields a simple polynomial time algorithm. The Zero-Inventory property states that in each period either there is no incoming inventory or there is no production.

1.3.1. Theorem (The Zero-Inventory property).
For each instance of ELS, there is an optimal solution such that for each period t $(t=1,\ldots,T)$ *the following holds:* $I_{t-1}x_t=0$.

This can be proved with an easy exchange argument. A more general way is to use the following network flow formulation of ELS:

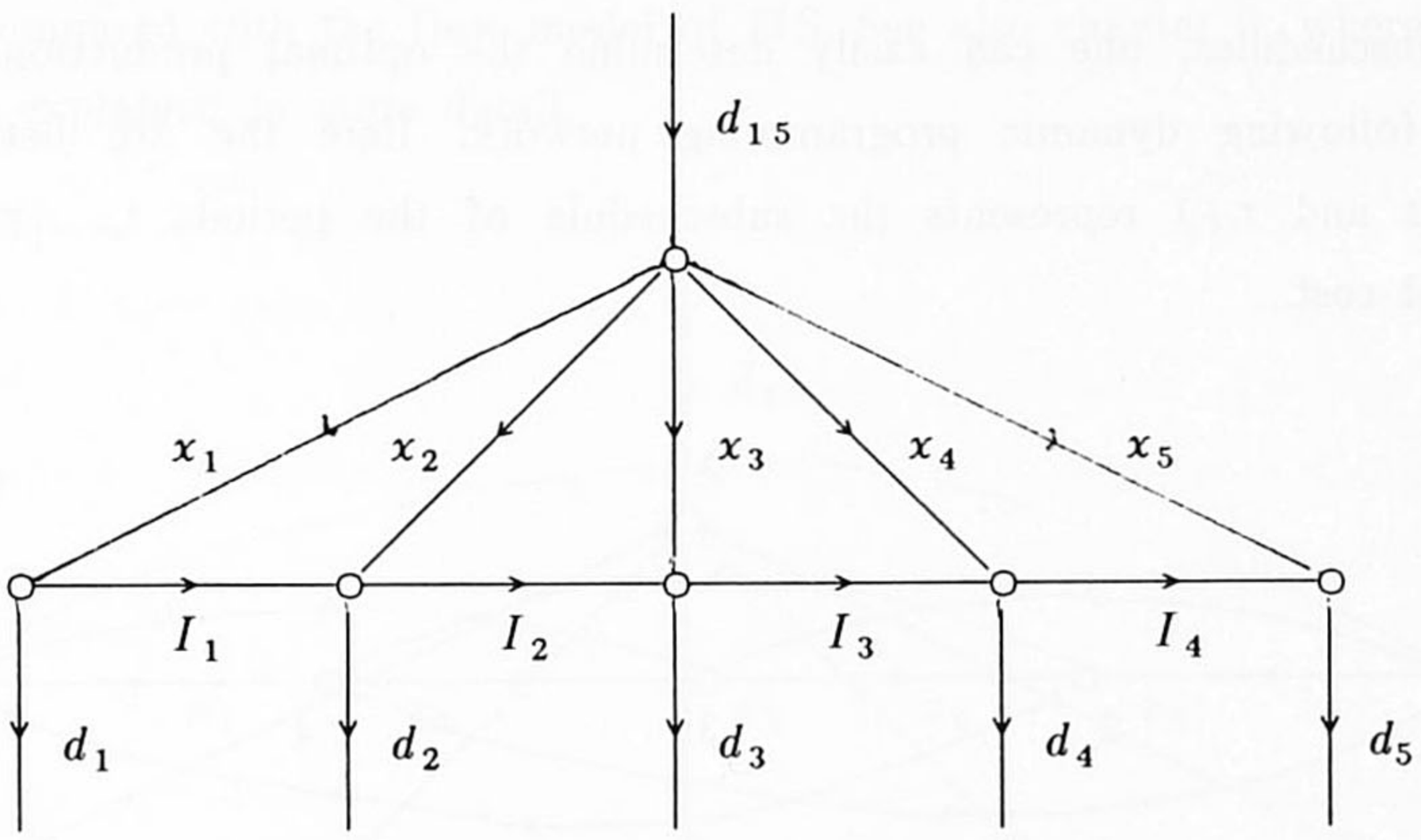


Figure 1.2. *Flow formulation of ELS.*

The arcs of the production variables have fixed costs consisting of the set-up costs, next to the unit production costs. This type of network, which is characterized by the fact that there is only one arc with incoming flow, has an arborescent optimal flow, i.e., a flow without cycles. This result is due to Veinott [50]. However, it is not hard to see this result, since non-arborescent flows can be changed into arborescent ones without increasing costs, though. This result implies the zero-inventory property immediately, since for each node only one incoming arc contains positive flow in an arborescent flow, i.e., $I_{t-1}x_t=0$ $(1<t\leq T)$. The zero-inventory property is also valid for concave production and holding costs, by the same argument.

The zero-inventory property can be generalized to hold for extensions of ELS, like the backlogging model and the model with start-up costs. Since this is the topic of Chapter 3 we refer to that chapter for the details.

The main virtue of the Zero-Inventory property is that one can characterize feasible solutions, as being built up of a set of subschedules $S_{t\tau}$, where production of the demands of the periods $t,\ldots,\tau$ takes place in period t. Thus a subplan consists of a smallest set of periods with no incoming inventory ($I_{t-1}=0$) and no leaving inventory ($I_\tau=0$).

Calculating the minimal cost of a subschedule $S_{t\tau}$ for ELS is trivial, since t is the only production period. This cost is $f_t+c_t d_{t,\tau}$. Given the costs of all $O(T^2)$ subschedules, one can easily determine the optimal production schedule by the following dynamic programming network. Here the arc between the vertices t and $\tau+1$ represents the subschedule of the periods $t,\ldots,\tau$ with its associated cost.

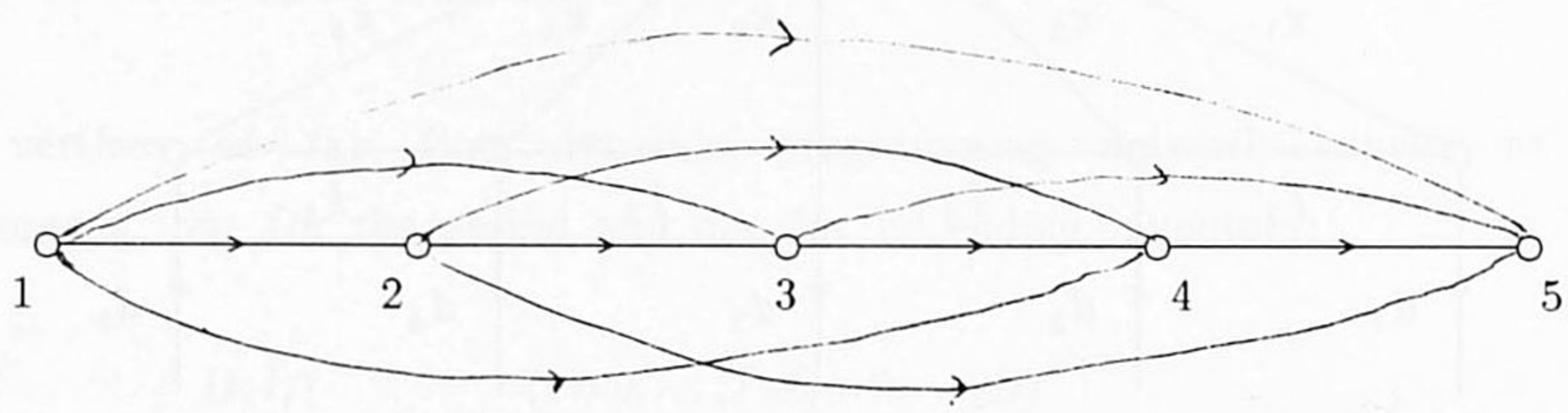


Figure 1.3. *Dynamic programming network for ELS* ($T=4$).

Since the number of arcs in the network is of order T^2, this is also the complexity of the dynamic program. It is the original algorithm of Wagner and Whitin [54], i.e., the production and holding costs are linear in the amount of production and stock, respectively. However, contrary to the problem of Wagner and Whitin, these costs may vary over the planning horizon. The concave cost version was shown to fit in the above model in an addendum of Wagner [53]. The forward and backward dynamic programming recursions for ELS are the following.

$$F(0) := 0; \qquad F(t) = \min_{\tau \le t}\{F(\tau-1) + f_\tau + c_\tau d_{\tau,t-1}\} \qquad (t = 1,\ldots,T)$$

$$B(T+1) := 0; \; B(t) = f_t + \min_{\tau > t}\{B(\tau) + c_t d_{t,\tau-1}\} \qquad (t = T,\ldots,1)$$

The notion of subschedules can be generalized to capacitated economic lot sizing problems, see Bitran and Yanasse [7]. In that case they lead to the same dynamic programing network as defined here. However, in that case it is harder to calculate the costs of the arcs.

Since ELSS differs only from ELS with respect to the fixed costs, it is easily shown that the zero-inventory property also holds for ELSS. Therefore, we will just provide the flow model for ELSS, where some extra fixed cost arcs are added, compared with the flow model of ELS. See also chapter 6, where the flow model is explained in more detail.

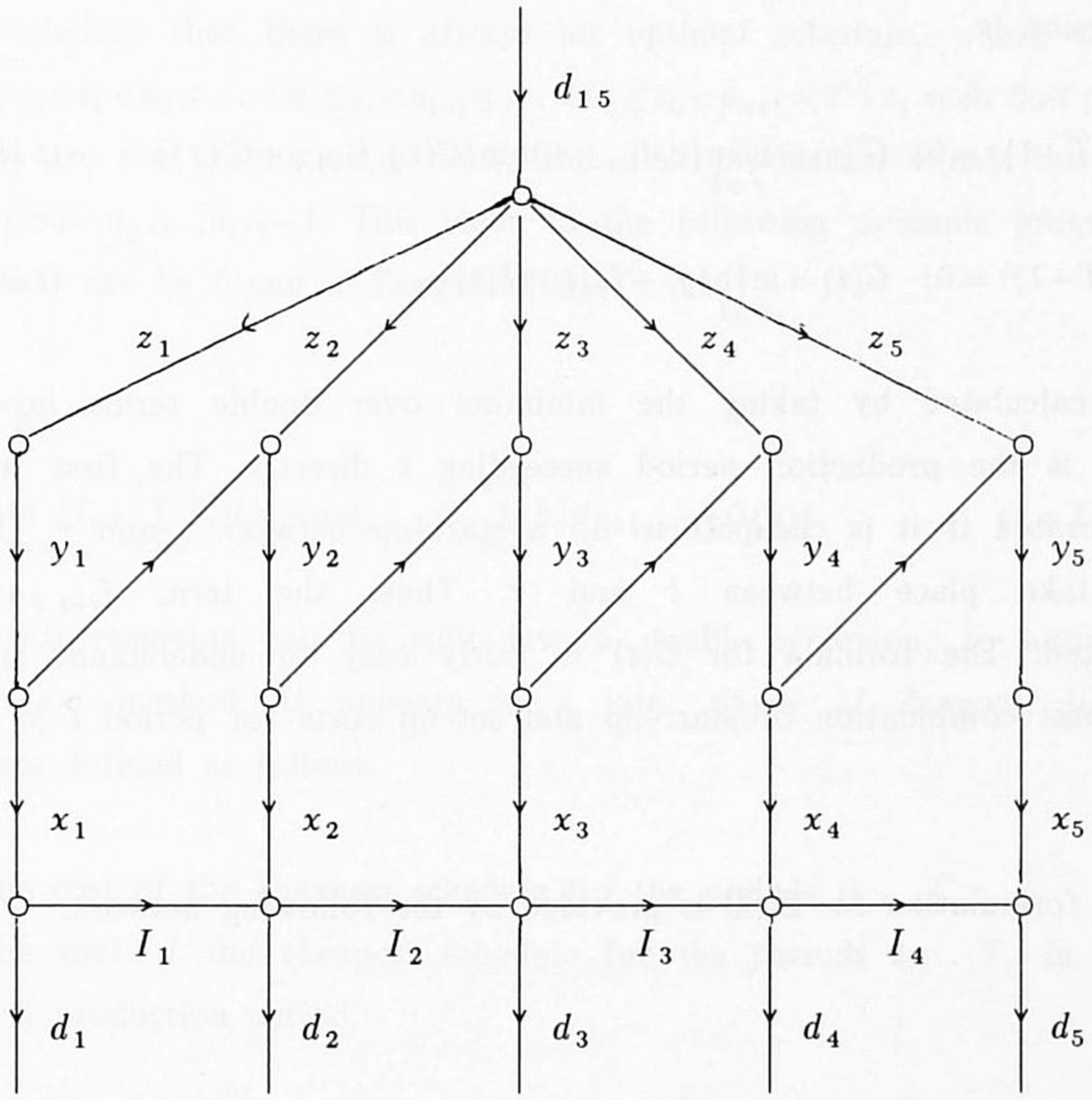


Figure 1.4. *Flow model for ELSS.*

The dynamic programming recursion consists of two types of variables, in order to deal with the more general fixed costs.

$G(t)$: The minimum cost of producing the demands of the periods $t,\ldots,T$, in these periods.

Clearly, in ELSS it might be profitable to perform a start-up in a period previous to t, and to set-up in all periods in between. These costs are taken into account in the variables $G(t)$, but not in the following dynamic programming variables.

$G'(t)$: The minimum cost for producing the demands of periods $t,\ldots,T$ in the same periods, but without start-up and set-up costs of the periods $\{1,\ldots,t\}$.

The recursion is

$$G'(T+1):=0;\quad G'(t)=\min_{\tau>t}\{c_t d_{t,\tau-1}+\min\{G(\tau),f_{t+1,\tau}+G'(\tau)\}\qquad (t=T,\ldots,1)$$

$$G(T+1):=0;\quad G(t)=\min_{\tau\le t}\{g_\tau+f_{\tau t}\}+G'(t)\qquad (t=T,\ldots,1)$$

$G'(t)$ is calculated by taking the minimum over double terms involving τ. Period τ is the production period succeeding t directly. The first term $G(\tau)$ is incorporated if it is cheapest to do a start-up between t and τ. Otherwise, set-ups take place between t and τ. Thus, the term $f_{t+1,\tau}+G'(\tau)$ is incorporated. The formula for $G(t)$ is fairly easy to understand, since only the cheapest combination of start-up and set-up costs for period t is added to $G'(t)$.

The flow formulation for ELSB is provided by the following network.

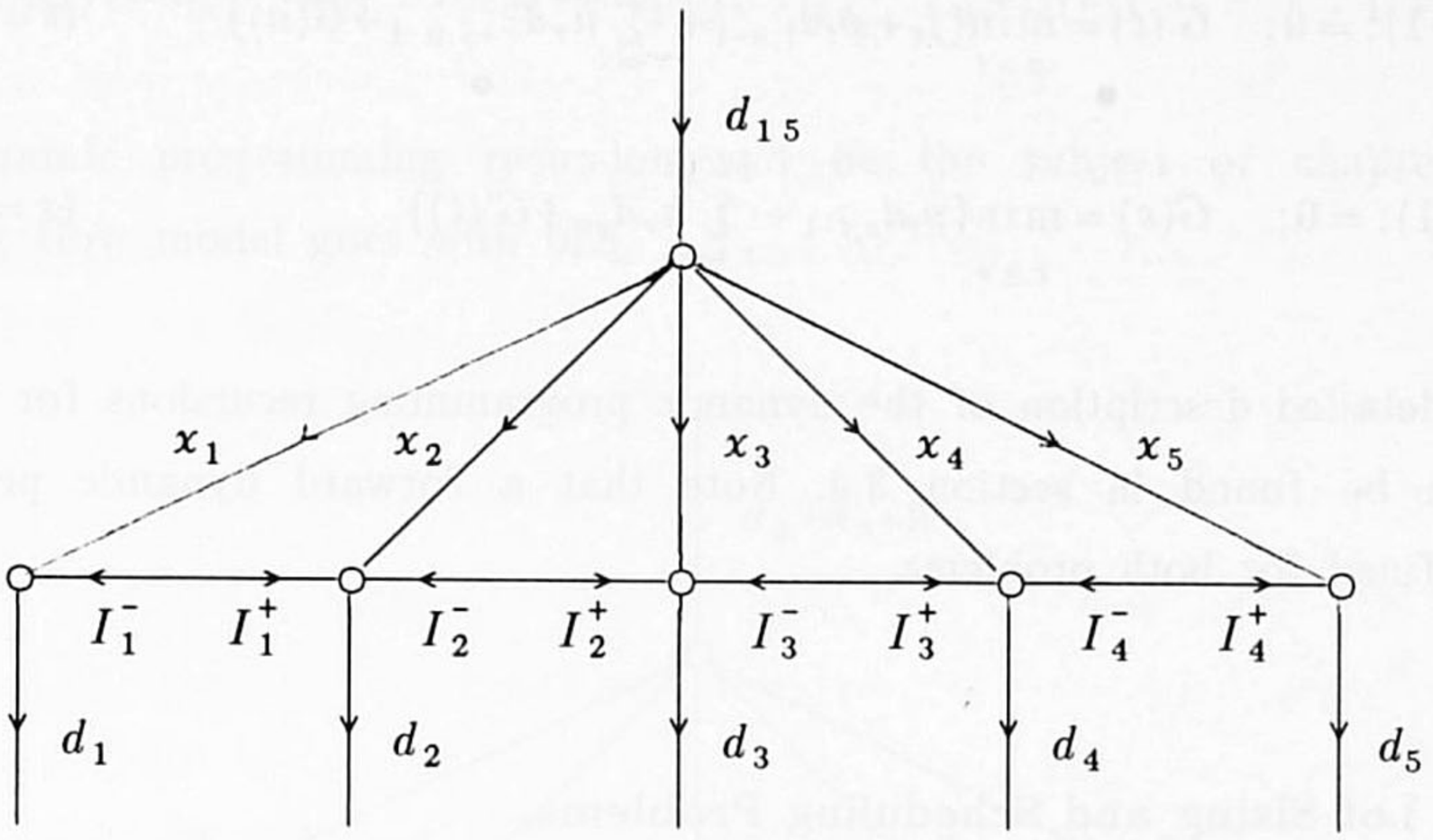


Figure 1.5. *Flow formulation for ELSB.*

From this flow formulation and the fact that there is an arborescent flow, one can easily deduce that there is always an optimal schedule, where there are periods $1 \le u_1 \le t_1 < u_2 \le \ldots < u_i \le t_i < u_{i+1} \le \ldots < u_n \le t_n < u_{n+1} = T+1$, such that periods t_i $(i=1,\ldots,n)$ are the production periods, and their production equals the demand of the periods $u_i,\ldots,u_{i+1}-1$. This leads to the following dynamic programming recursion that can be found in Zangwill [60].

$$G(T+1) := 0;$$

$$G(s) = \min_{s \le t < u} \{f_t + \sum_{\tau=s}^{t-1} h_\tau^- d_{s\tau} + p_t d_{s,u-1} + \sum_{\tau=t}^{u-1} h_\tau^+ d_{\tau+1,u-1} + G(u)\} \qquad (s = T,\ldots,1)$$

However, this recursion can be split into a double recursion, to allow for a faster solution method. It appears in a later paper of Zangwill [61]. The variables are defined as follows.

$G(t)$: The cost of the cheapest schedule for the periods $t,\ldots,T$.

$G'(t)$: The cost of the cheapest schedule for the periods $t,\ldots,T$, in which t is a production period.

The recursion is the following.

$$G'(T+1):=0; \quad G'(t)=\min_{u>t}\{f_t+p_t d_{t,u-1}+\sum_{\tau=t}^{u-1} h_\tau^+ d_{\tau+1,u-1}+G(u)\} \qquad (t=T,\ldots,1)$$

$$G(T+1):=0; \quad G(s)=\min_{t\geq s}\{p_t d_{s,t-1}+\sum_{\tau=s}^{t-1} h_\tau^- d_{s\tau}+G'(t)\} \qquad (s=T,\ldots,1)$$

A more detailed description of the dynamic programming recursions for ELSS and ELSB can be found in section 3.4. Note that a forward dynamic program is easily defined for both problems,

Discrete Lot Sizing and Scheduling Problems.

The dynamic programming algorithm for DLS is an adapted version of the first Dynamic Program given in this section for ELS. Although that dynamic program is not efficient for ELS, it is for DLS, because of the fact that production and demand in each period are binary. This reduces the number of states in the network to $O(DT)$, where D is the cumulative demand of all periods. The number of arcs leaving each node is reduced to two, since there are only two production decisions that can be made. However, there is one small problem, since the fixed cost structure differs from that of ELS. This problem is solved by replicating each node once, and introducing an additional parameter p.

$$(t, I_t, p) \qquad (t=1,\ldots,T+1; I_t=0,\ldots,D; p=0,1)$$

Here, p indicates whether production takes place in period t or not. The number of arcs leaving (t, I_t, p) is at most two and the number of vertices is $O(DT)$, and therefore the complexity bound for solving this dynamic program is $O(DT)$.

However, this is not the type of dynamic program with which we will work. The type of dynamic program that we consider only reviews schedules in which production takes place in periods, where the starting inventory is zero. Then the following dynamic programming recursion can be defined.

$$G(T+1):=0; \quad G(t) := g_t + \min_{\tau > t}\{c_{t,t+d_{t,\tau-1}-1} + G(\tau)\} \qquad (t=T,\ldots,1)$$

This dynamic programming recursion will be the subject of chapter 4. The following flow model goes with DLS.

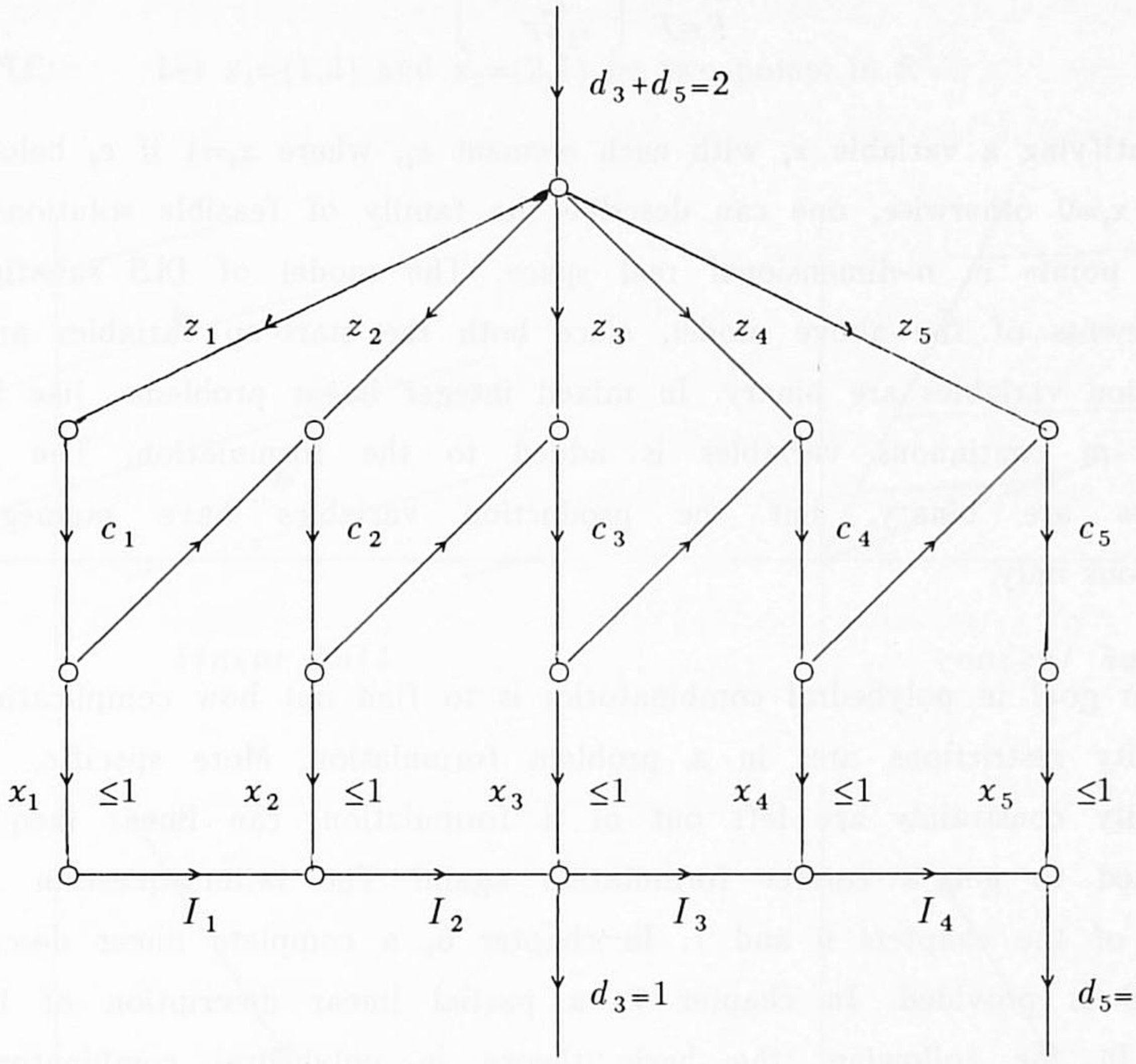


Figure 1.6. *Flow model for DLS* ($T=5$).

In this flow model, the arcs corresponding to the z_t and the c_t are the fixed cost arcs. Note that the production x_t and the costs of production c_t are separated.

1.4. Basic techniques in polyhedral theory

A typical formulation of a combinatorial problem is the following. A finite ground set E is given which consists of n elements $e_1, e_2, \ldots, e_n$. With each

element a weight w_i $(i=1,\ldots,n)$ is associated. A family of subsets of E is identified implicitly, to be the set of feasible solutions. This family is denoted by $\mathcal{F}$. The aim is to find the feasible solution which contains the elements with the smallest cumulative weight:

$$\min_{F\in\mathcal{F}} \left\{ \sum_{e_i\in F} w_i \right\}$$

By identifying a variable x_i with each element e_i, where $x_i=1$ if e_i belongs to F and $x_i=0$ otherwise, one can describe the family of feasible solutions as a set of points in n-dimensional real space. The model of DLS satisfies the requirements of the above model, since both the start-up variables and the production variables are binary. In mixed integer linear problems, like ELS, a set of m continuous variables is added to the formulation. The set-up variables are binary, but the production variables have nonnegativity restrictions only.

A major goal in polyhedral combinatorics is to find out how complicating the integrality restrictions are, in a problem formulation. More specific, if the integrality constraints are left out of a formulation, can linear inequalities be added to get a correct formulation again? The latter question is the subject of the chapters 6 and 7. In chapter 6, a complete linear description of ELSS is provided. In chapter 7, a partial linear description of DLS is given. In the following, the basic theory in polyhedral combinatorics is reviewed. For a more extensive treatment on the subject, see Nemhauser and Wolsey [37], and Schrijver [49].

Important concepts in the theory of linear descriptions are *linear combinations* and *convex combinations*. We start with the definitions.

Suppose that we are given a set of k points in n-dimensional space $\mathbb{R}^n$, say $x_1, x_2, \ldots, x_k$. The *linear hull* of these points is the set $\{\sum_{i=1}^k \alpha_i x_i | \alpha_i \in \mathbb{R}\}$. The set of points is called linearly independent if and only if $\sum_{i=1}^k \alpha_i x_i = 0$ implies $\alpha_i=0$ for all $i\in\{1,\ldots,k\}$. The *conical hull* of these points is the set $\{\sum_{i=1}^k \alpha_i x_i | \alpha_i \geq 0\}$. The *affine hull* of these points is the set $\{\sum_{i=1}^k \alpha_i x_i | \sum_{i=1}^k \alpha_i = 1\}$. The set is called affinely independent if and only if the

all-zero vector does not belong to the affine hull (unless one of the points equals the all-zero vector, of course), i.e., the equations $\sum_{i=1}^{k}\alpha_i x_i = 0$ and $\sum_{i=1}^{k}\alpha_i = 1$ have no solution. The *convex hull* of the points is $\{\sum_{i=1}^{k}\alpha_i x_i | \sum_{i=1}^{k}\alpha_i = 1 \text{ and } \alpha_i \geq 0\}$. The various types of hulls, defined here, are illustrated in the following example.

EXAMPLE: Let $x_1=(1,3)$ and $x_2=(2,1)$ be two points in $\mathbb{R}^2$.

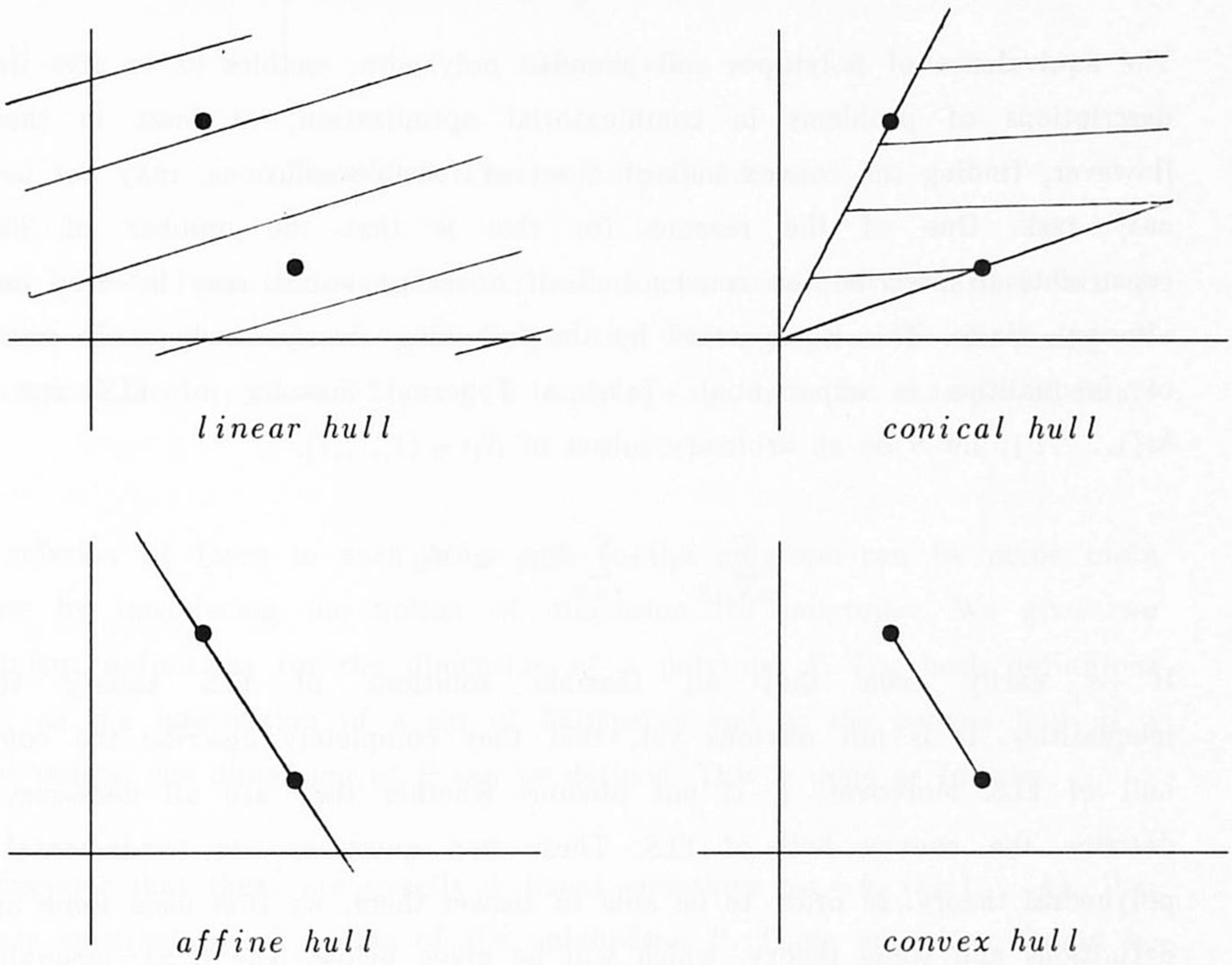


Figure 1.7. *The four types of hulls.*

The convex hull of a finite set of points in $\mathbb{R}^n$ is called a *polytope*. Thus, by the definition of problems in combinatorial optimization, we can regard the convex hull of the set of feasible solutions of a problem as a polytope. The importance of this definition lies in the close relation of the concept of polytopes to the concept of polyhedra, which will be given below.

A half-space in n-dimensional real space is the set of points satisfying an inequality of the type $ax \leq b$ where a is an n-vector and b is a scalar. The intersection of a finite set of half-spaces is a *polyhedron*.

Minkowsky [35] and Weyl [56] proved that polytopes and bounded polyhedra are equivalent structures. Motzkin [36] proved a more general version, namely that a polyhedron is the linear sum of a polytope and a conical hull. Due to these results, combinatorial optimization problems are related to polyhedra.

The equivalence of polytopes and bounded polyhedra, enables us to give linear descriptions of problems in combinatorial optimization, at least in theory. However, finding the convex hull of a set of feasible solutions, may not be an easy task. One of the reasons for this is that the number of linear constraints to describe the convex hull of a set of points may be very large, although finite. This is suggested by the following example, where the number of inequalities is exponential. Take a T-period instance of ELS and let $l \in \{1,\ldots,T\}$; let S be an arbitrary subset of $N_l := \{1,\ldots,l\}$.

$$\sum_{t \in N_l \setminus S} x_t + \sum_{t \in S} d_{tl} y_t \geq d_{1l}$$

It is easily seen that all feasible solutions of ELS satisfy these inequalities. It is not obvious yet, that they completely describe the convex hull of ELS. Moreover, it is not obvious whether they are all necessary to describe the convex hull of ELS. These two questions are fundamental in polyhedral theory. In order to be able to answer them, we first need some more definitions and some theory, which will be given below. The (l,S)-inequalities defined above will serve as an example.

Given an n-vector a and a scalar b consider the inequality $ax \leq b$. This inequality is called valid if every feasible solution x satisfies the inequality. Actually, for a given n-vector a, the values for b for which a valid inequality arises are easily seen to be bounded from below by the value $b_{min} := \max\{ax \mid x \in \mathcal{F}\}$. Therefore, this is the best right-hand side that one can get. The convex hull of the set of feasible solutions which satisfies $ax = b_{min}$ is called a face of the polyhedron. Note that a face is also a polyhedron.

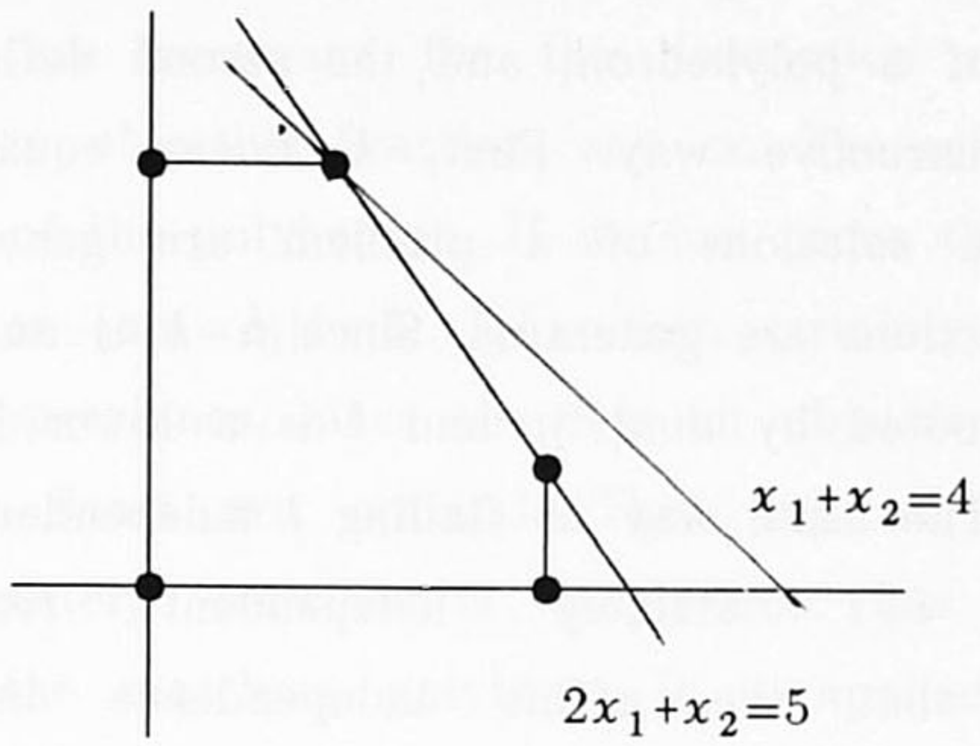


Figure 1.8. *Faces of a polyhedron.*

As can be seen from the example, one face can be contained in another face. If so, then there is a set of valid inequalities which implies the inequality of the original face: $x_2 \leq 3$ and $2x_1+x_2 \leq 5$ imply $x_1+x_2 \leq 4$, in the example of figure 1.7.

The relation of faces to each other and to the polytope can be made more precise by introducing the notion of dimension for polytopes. We give two equivalent definitions for the dimension of a polytope P. For both definitions of P, as the intersection of a set of halfspaces and as the convex hull of a set of points, the dimension of P can be defined. This is done as follows.

1) Suppose that there are exactly k linear equations $a_i x = b_i$ $(i=1,\ldots,k)$, that are satisfied by all points of the polyhedron P. These equations should be linearly independent, which means that the vectors a_i $(i=1,\ldots,k)$ should be linearly independent, Then the dimension of P, denoted by $dim(P)$, is $n-k$.

2) Let there be exactly l linearly independent directions d_j $(j=1,\ldots,l)$, where a direction is the difference of two points in P, i.e., $d_j = p_j - q_j$ where p_j and q_j are the two points in P. Then the dimension of P is l.

The determination of the dimension of a certain polyhedron usually uses both definitions. The first definition provides an upper bound in a constructive way on the dimension of a polyhedron, and the second definition provides a lower bound in a constructive way. First, k linear equations which are satisfied by all feasible solutions of a problem are generated; Second, l linearly independent directions are generated. Since $n-k$ is an upper bound on the dimension of P, denoted by $dim(P)$, and l is a lower bound, $dim(P)$ is determined, if $k+l=n$. The usual way of finding l independent directions in P is by identyfying $l+1$ affinely independent feasible solutions $p_1, p_2, \ldots, p_l, p_{l+1}$. Note that the affine independence implies that the directions $p_j - p_{l+1}$ $(j=1,\ldots,l)$ are linearly independent. However, although used more frequently, the latter is usually more complicated, since it is more restrictive in its choices of directions. This can be seen from the following simple example.

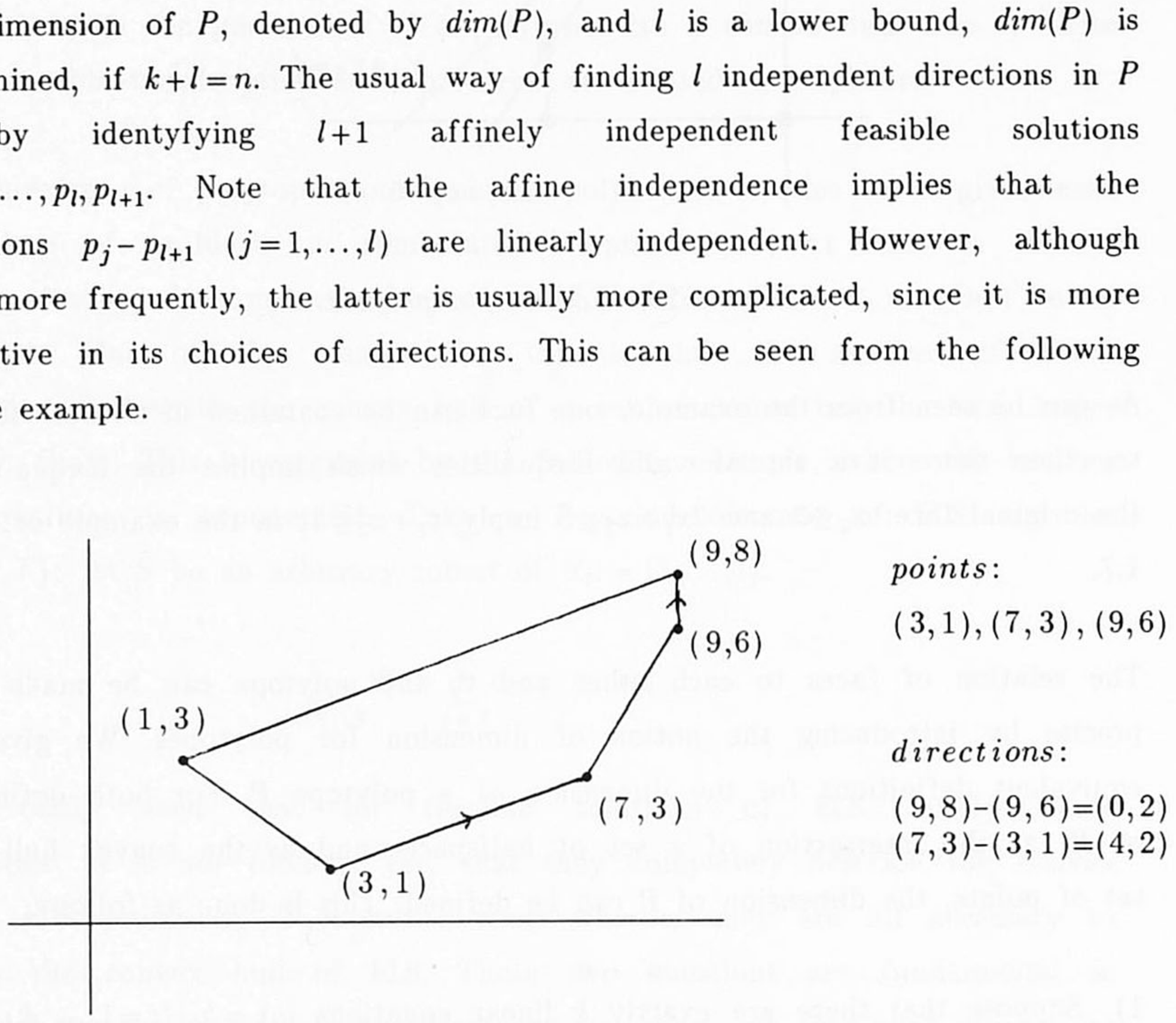


Figure 1.9. *Affine and linear independence in a polytope.*

In chapters 6 and 7 this concept is used to determine the dimension of faces. All proper faces of P, i.e., faces that are properly contained in P, have a dimension smaller than $dim(P)$. The proper faces with the highest dimension, $dim(P)-1$, are the facets. These are especially important, since a minimal set S of defining inequalities for P is such that, for each facet, there is exactly one element in S, which is satisfied as equality by all the points of the facet.

Now suppose that we have a set of valid inequalities $\{a_i x \leq b_i | i=1,\ldots,k\}$ for which we want to prove that they contain all facet-defining ones, i.e., all inequalities that are necessary for the description of P. This is done by taking an arbitrary objective function, say cx. The set of optimal solutions M defines a face F of the polyhedron. If we can prove that F is contained in one of the hyperplanes $\{a_i x = b_i | i=1,\ldots,k\}$, then certainly each facet is contained in one of the hyperplanes, since there is an objective function for which all the points in a facet are optimal. This way of proving that a set of inequalities completely describes a polyhedron has first been applied by Lovász [30] on the matching polytope. It is applied in chapter 6, on the economic lot sizing problem with start-up costs. Here we give such a proof for the inequalities of ELS, the $(l,S)-$inequalities. These inequalities have already been described earlier in this section. We will now prove that they contain all facet-defining inequalities, together with the inequalities in the formulation of ELS.

Consider the set of optimal solutions M to the objective function $fy+cx$. The components of c may be taken nonnegative by adding the right multiple of the equation $\sum_{t=1}^{T} x_t = d_{1T}$ to the objective function. Thus, $c_t \geq 0$ for each $t \in \{1,\ldots,T\}$.

Case 1. Suppose that $f_t < 0$ for some period t in this objective function. Then the solutions in M all satisfy $y_t = 1$. Otherwise, such a solution can be improved with respect to the objective function by putting this variable to 1, without violating feasibility.

Case 2. Suppose that $f_t \geq 0$ for all t. Now define $l := \max\{t|$ for all $\tau \leq t$ $c_\tau > 0$ or $f_\tau > 0\}$. So $c_{l+1} = 0$ and $f_{l+1} = 0$ and $l+1$ is minimal with that property (of course, if no such period exists, then $l \equiv T$). Let S be the set $\{t \leq l | c_t = 0\}$. Then the set of optimal solutions M, satisfies the thus defined (l,S)-inequality as equality. This is seen as follows. Suppose that an optimal solution does not satisfy the (l,S)-inequality as equality:

$$\sum_{t \in N_l \setminus S} x_t + \sum_{t \in S} d_{tl} y_t > d_{1l} \qquad \text{for some optimal solution } (x,y) \in M$$

Define the period u to be the first period in S for which a set-up is performed, i.e., $y_u = 1$. If no such period exists we take $u = l+1$. Note that in this case $l < T$, since no overproduction is allowed in the model for ELS. The choice of u is such that production for the periods $\{u,\ldots,T\}$ can be done at no cost: if $u \leq l$, then $y_u = 1$ and $c_u = 0$; if $u = l+1$, then $f_u = c_u = 0$. If production takes place in periods $\{u+1,\ldots,l\}\backslash S$, this production can be transferred to u at a cost reduction, since for t in $\{u+1,\ldots,l\}\backslash S$ the production costs c_t are positive. If no production takes place in $\{u+1,\ldots,l\}\backslash S$, then the production in $\{1,\ldots,u-1\}\backslash S$ exceeds $\sum_{t=1}^{u-1} d_t$ the cumulative demand of these periods. Therefore, we can transfer the overproduction from the last production period in $\{1,\ldots,u-1\}\backslash S$ to u at a cost reduction as well. This finishes the proof.

Note that besides the (l,S)-inequalities, we only used the following constraints: $\sum_{t=1}^{T} x_t = d_{1T}$ and $y_t \leq 1$. Implicitly, we used the nonnegativity of the variables. Therefore, these are all the linear constraints necessary to describe the convex hull of ELS.

In the example above, most of the inequalities are facet-defining. The only exception are the inequalities for which period 1 is not in S. These inequalities can be left out of the description for ELS in case $d_1 > 0$, since then $y_1 = 1$.

Unfortunately, the (l,S)-inequalities are not uniquely described. This is a consequence of the fact that the polytope is not full-dimensional (the affine hull of the set of feasible solutions is not equal to $\mathbb{R}^{2T}$). This nasty property is not an exception but the rule for problems in combinatorial optimization. It has to do with the dimension of the polytope under consideration. We show this, using an earlier example, for the case of two dimensions. Suppose, in this example a third dimension is added. This dimension is denoted by the variable z and therefore we add the constraint $z = 0$ to the formulation of the problem in the example. Now the inequality $2x+y \leq 5$ still defines a facet of the polytope. However, it now defines a half-space in $\mathbb{R}^3$, instead of in $\mathbb{R}^2$. Adding multiples of z to the left-hand side of the inequality still maintains its property that the facet of the polytope is contained in the set of points satisfying $2x+y \leq 5$ as equality, thus $2x+y+\alpha z \leq 5$

defines the same facet, for any value of α. The planes $2x+y+\alpha z=5$ all contain the points (1,3,0) and (2,1,0). Moreover, there are no other planes containing these points except for the plane that contains the complete polytope, the plane $z=0$.

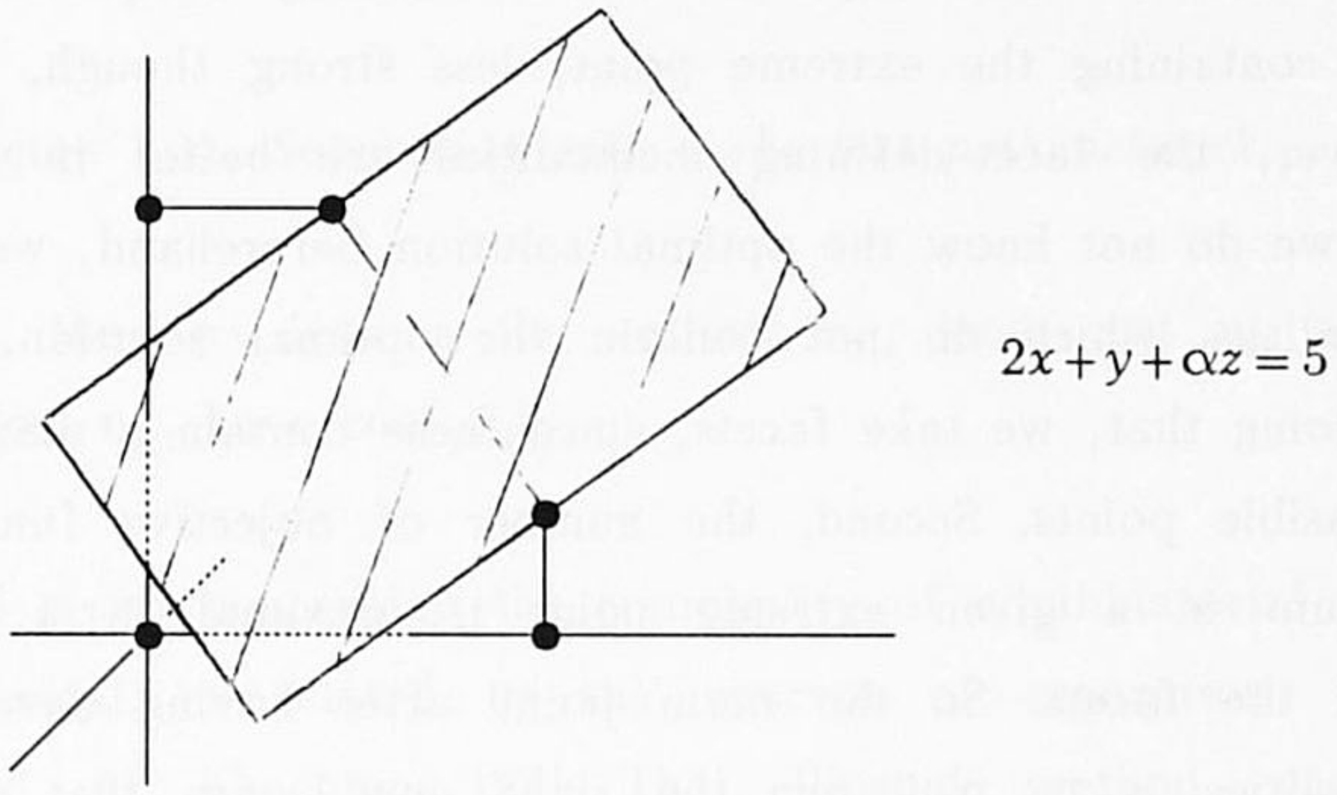


Figure 1.10. *Facet defining inequalities.*

If a polytope P is not full-dimensional, then the defining inequalities are not uniquely representable, i.e., suppose that all feasible solutions satisfy the equation $cx=d$, then the inequalities $ax\leq b$ and $(a+\alpha c)x\leq(b+\alpha d)$ contain the same subset of P. Thus, if this subset is a facet both inequalities define the same facet. In general, if a polytope P satisfies the equations $a_ix=b_i$ $(i=1,\ldots,k)$, then a facet is uniquely determined up to multiples of $a_ix=b_i$. In case of full-dimensional polytopes this implies that the facets are uniquely determined. In chapter 7, we will relax the model constraints of DLS to get a full-dimensional polyhedron.

Polyhedral methods in practice

In practice one is usually not interested in finding all valid or facet-defining inequalities, but in an optimal solution to the problem at hand. Since there is always an optimal solution which is an extreme point of the polytope, we are mainly interested in facet-defining inequalities that are satisfied at equality by the extreme optimal solution. Moreover, not even all of them may be necessary, since any point in n-dimensional space is uniquely

defined by n linearly independent equations, and there may be many more binding inequalities in an extreme point of a polytope. Consider, for instance, a pyramid, where the top has four binding inequalities. Here, three out of four inequalities are enough to define the top of the pyramid.

We do not need even only the facet defining inequalities, since other inequalities containing the extreme point, less strong though, may suffice, as well. However, the facet-defining inequalities are better in a double sense. First, since we do not know the optimal solution beforehand, we might generate valid inequalities which do not contain the optimal solution. To reduce the chance of doing that, we take facets, since these contain a maximal number of extreme feasible points. Second, the number of objective functions that find their optimum in a given extreme point is maximal in a description that contains all the facets. So the main point after having developed a set of valid inequalities is to pick out the right ones, ones that are violated by the optimal solution of a partial linear description. This is done by an algorithm that finds a separating inequality from a set of inequalities. Such an algorithm is called a separation algorithm.

For the (l,S)-inequalities a separation algorithm is easily described. This is due to the fact that they have a very compact representation. For a fixed l, all the inequalities of this type are contained in:

$$\sum_{t=1}^{l} \min\{x_t, d_{tl} y_t\} \geq d_{1l}$$

Therefore, to find an (l,S)-inequality which is violated by some vector (x,y) with fractional values in the component corresponding to the set-up variables, one just needs to determine the minimum of x_t and $d_{tl}y_t$, for each t in $\{1,\ldots,l\}$ for a fixed l. This leads to a shortest path problem on the following network.

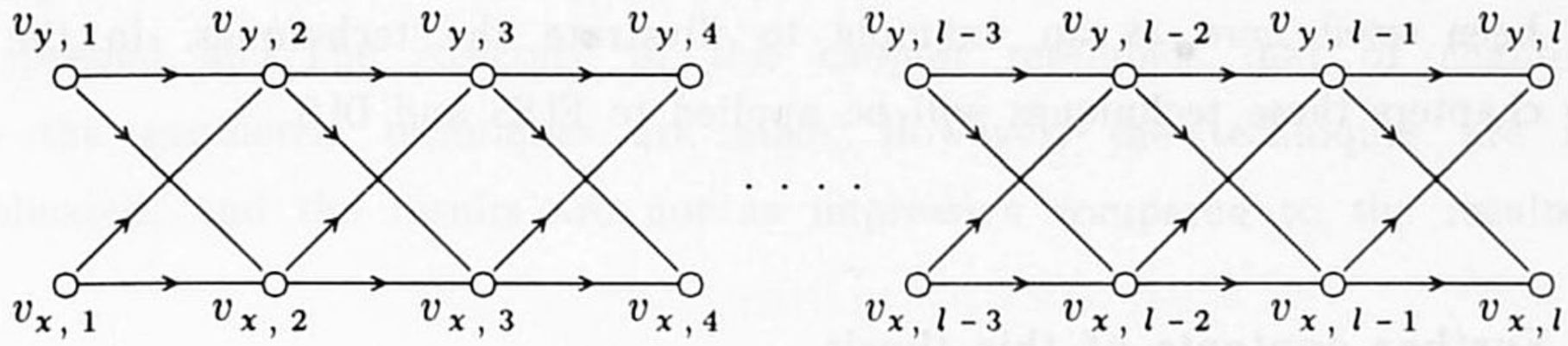


Figure 1.11. *Separation with a shortest path network.*

In this network, the arcs leaving the vertices $v_{x,t}$ have cost x_t, and the arcs leaving the vertices $v_{y,t}$ have cost $d_{tl}y_t$.

A very powerful result concerning the complexity of combinatorial optimization problems in general, is related to the concept of separation, due to the ellipsoid method of Khachian [25] The ellipsoid method was the first polynomial-time algorithm for linear programming. Although its practical implementation is worse than the simplex method, a nice feature of it is that it does not require a complete description of the list of defining inequalities, but a separation algorithm. This leads to the following result.

1.4.1. Theorem (Grötschel et al. [17]).
A problem is polynomially solvable if and only if for any point not in the convex hull of feasible solutions a separating hyperplane can be found in polynomial time.

An early result on separation for combinatorial problems is due to Gomory [16], who developed a finite algorithm to generate an inequality that separates a special point from a given partial linear description of a problem. However, the inequalities generated in that way, need not be strong in the sense that no feasible solution satisfies the inequality as equality. Another drawback is that the thus generated inequalities have many non-zero coefficients. This makes an LP-solver, based on the simplex method, slow and numerically instable.

So far, we have presented the basic theory of polyhedral combinatorics. ELS has been used here as an example to illustrate the techniques. In the final two chapters these techniques will be applied to ELSS and DLS.

1.5. Further contents of this thesis.

The remainder of this thesis can virtually be divided into three parts. The first part contains the chapters 2, 3 and 4. It deals with improved algorithms for ELS and DLS and their generalizations. The second part consists of chapter 5, only. Here, sensitivity analysis is performed on ELS, with the techniques developed in the preceding chapters. The third part contains the chapters 6 and 7. This part deals with linear descriptions of ELS (with start-up costs) and DLS, respectively.

In chapter 2, a historical review is given. Our main goal is to derive linear programming algorithms for ELS. Dual algorithms have our particular interest, because these may be a good starting point to perform sensitivity analysis. In the original model, we first had to add the inequalities of Barany et al. [3] to get a complete description. This led to an $O(T^2)$ algorithm. This algorithm has appeared in van Hoesel et al. [19]. Thereafter, we started to work on another linear programming formulation based on the facility location model. Although both algorithms were not very suitable for performing sensitivity analysis (at least not directly), the second algorithm could be implemented to run in $O(TlogT)$ time. This algorithm can be found in Wagelmans et al. [52]. Both algorithms are described in Chapter 2, together with some historical notes.

In chapter 3, the linear programming technique that was used for the facility location formulation of ELS in section 2, is reformulated as a geometric technique that can be used to speed up dynamic programs with linear costs. This technique is then used as a basis to speed up several extensions of ELS, like the case with start-up costs and the case with backlogging. Some computational results are given in this chapter as well.

In chapter 4, the dynamic programming algorithms for the three variants of DLS are speeded up. The structure of this chapter resembles that of chapter 3, since the geometric techniques are alike. However, the techniques are more complicated, and the results are not as impressive compared to the results for ELS.

In chapter 5, we derive results concerning our main goal, a sensitivity analysis for the Economic Lot Sizing problem. Here, the parameters are varied over certain ranges, one by one. It turns out that there is a striking difference between the various parameters of ELS and also between raising and lowering a single parameter, with respect to computational complexity.

In chapter 6, the first polyhedral results are derived. A complete linear description of the convex hull of ELSS is provided in this chapter. The facet-defining inequalities are determined, and a separation algorithm is described. It is work that relies highly on earlier work of Barany et al. [3] and [4], and Wolsey [57].

In chapter 7, partial results on the linear description of DLS are proved. Here, it becomes clear that binary programs can have very complicated structures with respect to their facet-defining inequalities. Finally, a more compact reformulation, similar to the uncapacitated facility location reformulation of ELS, is shown to give a complete description.

Chapter 2

LINEAR PROGRAMMING ALGORITHMS FOR THE ECONOMIC LOT SIZING PROBLEM

2.1. Introduction

In their paper on the "Dynamic version of the economic lot size model" Wagner and Whitin suggest an elegant solution method for the Economic Lot Sizing problem (ELS), which still stands as a classical application of Dynamic Programming. However, despite the relative simplicity of this algorithm, its running time seemed improvable: for problems with a planning horizon of T periods the complexity of the algorithm is $O(T^2)$. Actually, it is even $\Omega(T^2)$. Therefore, in the past three decades much effort has been put in lowering the computational burden of the dynamic programming algorithm. Evans [13] describes an efficient implementation of the algorithm and Saydam and McKnew [47] use the planning horizon theorem of Wagner and Whitin (see chapter 3, section 3) to speed up the algorithm. More sophisticated versions of the planning horizon theorem have been developed by Zabel [59], Eppen, Gould and Pashigian [10], and Lundin and Morton [31]. Nevertheless, none of the above ideas has evolved in an algorithm with improved worst-case running time, which led Bitran, Magnanti and Yanasse [6] to state that "The procedure of Wagner and Whitin is so efficient that the problem is often viewed as having been solved".

The main goal of this chapter is to develop a linear programming algorithm for ELS with a running time of $O(T\log T)$. Moreover, if the instances of ELS have a special cost structure (like the costs described by Wagner and Whitin), this running time can be improved to $O(T)$. At first glance it seems illogical to try to improve the dynamic programming algorithm by solving a linear programming formulation of ELS, since usually linear programming algorithms

have a poor running time compared with special purpose algorithms. Certainly, their running time almost never improves the running time of special purpose algorithms. However, when this research started the goal was not to develop a fast algorithm for ELS, but to perform a sensitivity analysis on the various parameters in the model of ELS. This idea had its foundations in the positive experiences with a comparable result based on the primal-dual algorithm of Edmonds [9] for the matching problem, and the fact that a complete linear description of ELS was available through the (l,S)-inequalities of Barany, Van Roy and Wolsey [4].

The dual linear programming algorithm, which uses the (l,S)-inequalities, was developed by van Hoesel, Kolen and Wagelmans [20]. It acts in a greedy fashion, in the sense that it selects the most profitable non-basic variable, and then this variable is increased as much as possible. A nice by-product of this algorithm was a constructive proof that the (l,S)-inequalities describe the convex hull of ELS. Unfortunately, the sensitivity analysis with this algorithm turned out to give rather unsatisfactory results, even if the set of constraints were decreased to a polynomial number, by adding some variables. Both models are provided in section 2.2. Therefore, a second formulation of the problem was tried. In this formulation the production variables are split. It is a special case of the Uncapacitated Facility Location problem (UFL), see chapter 1, and it has been developed by Krarup and Bilde [27]. They also give a dual algorithm to solve the linear programming relaxation of this model. This algorithm treats the variables in a forward order, i.e., in the order in which the periods appear. It also fixes these variables greedily. However, if the variables are treated in a backward fashion, and when these variables are not fixed in one iteration, the linear program can be solved in $O(T\log T)$ time. This backward algorithm has been implemented by Wagelmans, van Hoesel and Kolen [52]. Due to the close relation of the UFL-model and the dynamic programming formulation of ELS, an algorithm that solves the backward dynamic programming recursion in the same time bound could be developed quite soon afterwards. However, the linear programming algorithms are discussed in this chapter, only.

In section 2.2, the dual algorithm for the linear programming relaxation of the model in the original variables (the set-up and the production variables) is described. Moreover, it is shown how this model can be formulated in a polynomial number of constraints and variables, by introducing some extra variables. In section 2.3 the UFL-formulation is described, and the $O(T\log T)$ backward algorithm to solve its linear programming relaxation is developed. This chapter is finished with some comments in section 2.4.

2.2. A linear programming algorithm for ELS, with the (l,S)-inequalities.

The original formulation as given in chapter 1 is:

$$\text{(ELS)} \qquad \min \sum_{t=1}^{T} (f_t y_t + c_t x_t) \qquad (2.2.1)$$

$$\text{s.t.} \qquad \sum_{t=1}^{T} x_t = d_{1T} \qquad (2.2.2)$$

$$\sum_{\tau=1}^{t} x_\tau \geq d_{1t} \qquad (1 \leq t \leq T-1) \qquad (2.2.3)$$

$$x_t \leq d_{tT} y_t \qquad (1 \leq t \leq T) \qquad (2.2.4)$$

$$y_t \in \{0,1\} \qquad (1 \leq t \leq T) \qquad (2.2.5)$$

$$x_t \geq 0 \qquad (1 \leq t \leq T) \qquad (2.2.6)$$

The following inequalities, the (l,S)-inequalities are now added to this formulation. The validity of these inequalities has already been shown in chapter 1. For a given l, we define N_l as $\{1,\ldots,l\}$. Now let l be an arbitrary period in $\{1,\ldots,T\}$, and let S be an arbitrary subset of N_l.

$$\sum_{t \in N_l \setminus S} x_t + \sum_{t \in S} d_{tl} y_t \geq d_{1l}$$

Adding these inequalities, and relaxing the integrality constraints yields the following formulation, denoted by ELS-lS.

$$\text{(ELS-}lS\text{)} \qquad \min \sum_{t=1}^{T} (f_t y_t + c_t x_t) \tag{2.2.7}$$

$$\text{s.t.} \qquad \sum_{t=1}^{T} x_t = d_{1T} \tag{2.2.8}$$

$$\sum_{t \in N_l \setminus S} x_t + \sum_{t \in S} d_{tl} y_t \geq d_{1l} \qquad (1 \leq l \leq T; S \subseteq N_l) \tag{2.2.9}$$

$$y_t \leq 1 \qquad (1 \leq t \leq T) \tag{2.2.10}$$

$$y_t \geq 0 \qquad (1 \leq t \leq T) \tag{2.2.11}$$

$$x_t \geq 0 \qquad (1 \leq t \leq T) \tag{2.2.12}$$

Note that $\sum_{\tau=1}^{t} x_\tau \geq d_{1t}$ $(t=1,\ldots,T)$ and $x_t \leq d_{tT} y_t$ $(t=1,\ldots,T)$ can easily be derived from the given constraints above: the first type follows by taking the (l,S)-inequality with $l=t$ and S is empty; The second type follows by subtracting the (l,S)-inequality, where $l=T$ and $S=\{t\}$ from 2.2.8.

The corresponding dual linear program (DELS-lS) is determined as follows.

$$\text{(DELS-}lS\text{)} \qquad \max \left(d_{1T}\mu + \sum_{t=1}^{T} \nu_t + \sum_{l=1}^{T} \sum_{S \subset N_l} d_{1l} \sigma_{lS} \right)$$

$$\text{s.t.} \qquad \mu + \sum_{\substack{l \in N_T,\ S \subset N_l \\ t \in N_l \setminus S}} \sigma_{lS} \leq c_t \qquad (1 \leq t \leq T) \tag{2.2.13}$$

$$\nu_t + \sum_{\substack{l \in N_T,\ S \subset N_l \\ t \in S}} d_{tl} \sigma_{lS} \leq f_t \qquad (1 \leq t \leq T) \tag{2.2.14}$$

$$\nu_t \leq 0 \qquad (1 \leq t \leq T) \tag{2.2.15}$$

$$\sigma_{lS} \geq 0 \qquad (1 \leq l \leq T; S \subseteq N_l) \tag{2.2.16}$$

Here, the production and the set-up costs are allowed to be negative. Therefore, the first thing to do is to create a feasible solution to DELS-lS. This is done by inspection. The variable μ is set to $\min\{c_t | t=1,\ldots,T\}$ and each variable ν_t $(t=1,\ldots,T)$ is set to $\min\{0, f_t\}$. Then the variables σ_{lS} $(l=1,\ldots,T; S \subseteq N_l)$ can be set to zero. This starting solution is easily checked to be dual feasible. The gaps between the left-hand side and the right-hand

side in the constraints 2.2.13 and 2.2.14 are the reduced costs. Thus, for a period t the gap in the corresponding constraint of type 2.2.13 is called the reduced production cost of t, denoted by c_t^r, and the gap in the corresponding constraint of type 2.2.14 is called the reduced set-up cost of t, denoted by f_t^r.

The dual algorithm now proceeds to determine the successive values of the dual variables σ_{lS} $(l=1,\ldots,T;S\subseteq N_l)$ in a greedy manner, i.e., a variable with the highest profit with respect to the objective function, is chosen and this variable is increased as much as possible. The variables σ_{lS} have coefficient d_{1l}. Therefore, l is chosen as large as possible. Since for each period $t\leq l$ either the reduced production or the reduced set-up costs are decreased when σ_{lS} is raised, a necessary condition for l is that for each period $t\in\{1,\ldots,l\}$, either the reduced production costs or the reduced set-up costs are positive. Therefore, we choose l maximal such that this condition is satisfied. Clearly, either $l=T$ or the period $l+1$ is the first period for which both the reduced production costs and the reduced set-up costs are zero.

The stopping criterion is directly related to the choice for l. Note that, if l is such that $d_{1l}=0$, then the increase of σ_{lS} for any $S\subseteq N_l$ will not increase the objective function, since its coefficient is d_{1l}. Therefore, the stopping criterion for the algorithm is precisely $d_{1l}=0$. In case the demand in the first period is positive, this stopping criterion is equivalent to $l=0$.

Finally, the choice of S is based on the reduced production costs of all periods in $\{1,\ldots,l\}$. The periods for which the reduced production costs are zero become elements of S. In fact, these are the periods that are obliged to be in S. If such a period were no element of S, then its reduced production cost would be reduced when σ_{lS} is increased and become negative, which makes the solution to DELS-lS infeasible.

Fortunately, from the choice of l it follows that for these periods the reduced set-up costs are positive, and therefore it is immediately clear that the value of σ_{lS} for this choice of S can be increased. Since we argued that the algorithm proceeds greedily, the value of σ_{lS} is fixed as high as

possible. For each period t in $\{1,\ldots,l\}$, there is a clear upper bound on the increase of σ_{lS}, depending on whether $t \in S$ or not. If t is not an element of S, then the reduced production costs define an upper bound on σ_{lS}, see 2.2.13. If t is an element of S, then the reduced set-up costs define an upper bound on $d_{tl}\sigma_{lS}$, see 2.2.14. Therefore, σ_{lS} is fixed to $\min\{\sigma_1,\sigma_2\}$, where $\sigma_1 = \min\{c_t^r | t \in N_l \backslash S\}$ and $\sigma_2 = \min\{f_t^r / d_{tl} | t \in S\}$.

For each period t that enters S, a period $l(t)$ is defined. It denotes the final period for which demand is produced in period t, if t is a production period.

At the beginning of the algorithm, for each period t which has reduced production costs zero, $l(t)$ is set to T. For each other period t, the value of $l(t)$ is set to zero. In general, $l(t)$ is defined to be the l in the "current" iteration, for those periods t for which the reduced production costs become zero, in the "current" iteration. This choice is induced by the fact that, if t is decided to be production period, then it is favourable to produce up till period l. First, more production is disadvantageous because production can take place at no (reduced) cost in period $l+1$ (regarding the current solution: the reduced costs for $l+1$ are zero). Second, less production is also disadvantageous, since that means that there is another production period in $\{t+1,\ldots,l\}$ and for this period either positive reduced set-up costs or positive reduced production costs must be paid.

EXAMPLE

t	1	2	3	4	5	6
d_t	3	2	1	2	1	
f_t^r	0	12	7	4	5	0
c_t^r	3	0	1	0	2	0

In this example l is chosen to be period 5 and S is chosen to be $\{2,4\}$. Now the upperbounds on the value of σ_{lS} are the following.

Period 1: $c_1^r = 3$; period 2: $f_2^r / d_{25} = 12/6 = 2$;
period 3: $c_3^r = 1$; period 4: $f_4^r / d_{45} = 4/3$;
period 5: $c_5^r = 2$.

Thus, the minimum is reached for period 3, and therefore σ_{lS} becomes 1. Moreover, since the reduced production costs of period 3 become zero $l(3)$ is defined to be period 5.

The primal algorithm is rather simple. First, for those periods t for which ν_t is negative, the set-up variable y_t is set to 1. Then it starts by defining $l+1$, where l is the final value in the dual algorithm, as the first production period. Note that for this value of l, $d_{1l}=0$ holds. Now consider a period t, which is identified as a production period. The production in this period equals the demand of the periods $t,\ldots,l(t)$. If period $l(t)$ is T, then the primal algorithm stops, otherwise period $l(t)+1$ is identified as a production period and so on. Note that the reduced production costs in each period t that is determined as a production period, are zero, at the end of the algorithm. This is due to the fact that $l(t)$ equals the value of l, for some iteration.

So far, a solution to DELS-*lS* and a solution to ELS-*lS* have been constructed. It remains to be shown that these two solutions are optimal. This is done by showing that the complementary slackness relations hold for both solutions.

The complementary slackness relations corresponding to the primal variables y_t and x_t $(t=1,\ldots,T)$ are: $y_t f_t^r=0$ and $x_t c_t^r=0$ $(t=1,\ldots,T)$, where f_t^r and c_t^r are the reduced set-up and production costs at the end of the dual algorithm. If $y_t=1$, then either $\nu_t<0$ or t is a production period. In the first case the reduced set-up costs are zero at the start of the algorithm, and this remains so, since the reduced costs are non-negative and non-increasing during the dual algorithm. With respect to the second case it suffices to notice that the production periods in the primal algorithm are chosen such that the reduced set-up costs and the reduced production costs are zero. Simultaneously, this implies that the complementary slackness relations with respect to the production variables are satisfied.

The complementary slackness relations with respect to the dual variables ν_t $(t=1,\ldots,T)$ are $\nu_t(1-y_t)=0$. These relations hold, since the primal algorithm assigns the value 1 to y_t whenever $\nu_t<0$.

The complementary slackness relations with respect to the variables σ_{lS} $(t=1,\ldots,T;S\subseteq N_l)$ are

$$\sigma_{lS}\left\{d_{1l} - \left[\sum_{t\in N_l\setminus S} x_t + \sum_{t\in S} d_{tl}y_t\right]\right\} = 0$$

Thus, it remains to be proved that if $\sigma_{lS}>0$, then the corresponding (l,S)-inequality is satisfied at equality. In order to do so the following lemma is used.

2.2.1. Lemma.

If $\sigma_{lS}>0$ for some choice of l and S, then for each $t\in N_l\setminus S$: $l(t)\leq l$; and for each $t\in S$: $l(t)\geq l$.

Proof. For any period t the value $l(t)$ is determined to be the value of l at the iteration, where the reduced production costs become zero. Thus, at the iteration where σ_{lS} is increased, the values $l(t)$ for $t\in S$ have been determined, and therefore their value is at least l. The values $l(t)$ for $t\in N_l\setminus S$ have not yet been determined. Therefore, either they remain zero, or they get a value of at most l.

□

Consider a choice of l and S for which σ_{lS} is positive. In the following, we prove that the corresponding (l,S)-inequality is satisfied at equality. First, consider the periods $t\in\{1,\ldots,l\}$, with $\nu_t<0$. Then the reduced set-up costs of t are zero throughout the dual algorithm. Therefore, throughout the dual algorithm, either $t\in N_l\setminus S$ or $l<t$, since as soon as the reduced production costs of t become zero, then l becomes smaller than t. Therefore, the variable y_t of such a period does not contribute to the left-hand side of the (l,S)-inequality.

Let t be the last production period in $\{1,\ldots,l\}$. The production periods in $\{1,\ldots,t-1\}$ are not in S, since otherwise, by lemma 2.2.1, t can not be a production period. The production in the periods $\{1,\ldots,t-1\}$ does not exceed $d_{1,t-1}$ which can easily be seen from the construction of the primal solution. Therefore, this is the contribution of the periods in $\{1,\ldots,t-1\}$ to the

left-hand side of the (l,S)-inequality. If $t \in N_l \backslash S$, then x_t is in the left-hand side of the (l,S)-inequality. By lemma 2.2.1 $l(t) \leq l$, and the production in t is at most d_{tl}. If $t \in S$, then y_t is in the left-hand side of the (l,S)-inequality, and $y_t = 1$. Its coefficient is d_{tl}. In both cases the contribution of t to the left-hand side is d_{tl}. Since t is the last production period in $\{1,\ldots,l\}$, the periods after t do not contribute to the left-hand side. In both cases the left-hand side of the (l,S)-inequality has value d_{1l}.

The complexity of the algorithm is easily determined. In each iteration the minimum of at most T numbers should be determined. Moreover, the number of iterations is $O(T)$. This follows from the fact that $l + |N_l \backslash S| \leq 2T$ at the start of the dual algorithm, and this value decreases in each iteration by at least one. The complexity of the algorithm is therefore $O(T^2)$.

Although, the complexity of the algorithm equals the complexity of the algorithm of Wagner and Whitin, it is surprising that this complexity can be reached with a linear programming based algorithm, where the linear program has an exponential number of inequalities. Actually, the latter is not quite true, since this number can be reduced by adding some variables in a very basic way. To do so, a more compact, non-linear, formulation of the (l,S)-inequalities will be given. For each period t consider the minimum of x_t and $d_{tl}y_t$. Then the complete set of (l,S)-inequalities for a fixed l can be described by the inequality

$$\sum_{t=1}^{l} \min\{x_t, d_{tl}y_t\} \geq d_{1l}$$

This inequality can be linearized by introducing the variables u_{tl} $(1 \leq t \leq l \leq T)$. These constitute a lower bound for both x_t and $d_{tl}y_t$. Thus, the formulation for ELS becomes the following.

$$\text{(ELS)} \qquad \min \sum_{t=1}^{T} (f_t y_t + c_t x_t) \tag{2.2.17}$$

$$\text{s.t.} \qquad \sum_{t=1}^{T} x_t = d_{1T} \tag{2.2.18}$$

$$\sum_{t=1}^{l} u_{tl} \geq d_{1l} \qquad (1 \leq l \leq T) \tag{2.2.19}$$

$$u_{tl} \leq d_{tl} y_t \qquad (1 \leq t \leq l \leq T) \tag{2.2.20}$$

$$0 \leq u_{tl} \leq x_t \qquad (1 \leq t \leq l \leq T) \tag{2.2.21}$$

$$y_t \geq 0 \qquad (1 \leq t \leq T) \tag{2.2.22}$$

This is called a compact formulation, since the number of variables and the number of linear constraints is polynomial in the size of the problem, the number of periods. However, it should be noted that this model is realized by adding $O(T^2)$ variables.

Finally, it should be mentioned that the close relation of both formulations enables us to adapt the dual algorithm and the primal algorithm for the formulation with the (l,S)-inequalities to the above formulation.

2.3. A linear programming algorithm for the uncapacitated facility location formulation of ELS.

The main goal in this section is to develop an algorithm that solves ELS in $O(T\log T)$ time. The linear programming formulation that plays an important role in the description of the algorithm is presented first. It is created from ELS by disaggregating the production variables x_t into variables $x_{t\tau}$ $(1 \leq t \leq \tau \leq T)$ which denote the quantity of the production in period t, used to satisfy demand in period τ. The splitting of the production variables leads to the following formulation, which is a scaled version of the so-called uncapacitated facility location formulation, as introduced by Krarup and Bilde [27]. See also section 1.2.

$$\text{(ELS-UFL)} \qquad \min \sum_{t=1}^{T} (f_t y_t + c_t(\sum_{\tau=t}^{T} x_{t\tau})) \tag{2.3.1}$$

$$\text{s.t.} \quad \sum_{t=1}^{\tau} x_{t\tau} = d_\tau \qquad (1 \le \tau \le T) \tag{2.3.2}$$

$$x_{t\tau} \le d_\tau y_t \qquad (1 \le t \le \tau \le T) \tag{2.3.3}$$

$$x_{t\tau} \ge 0 \qquad (1 \le t \le \tau \le T) \tag{2.3.4}$$

$$y_t \in \{0,1\} \qquad (1 \le t \le T) \tag{2.3.5}$$

To make the exposition in the following more transparent, the following simplifying assumptions are made. For each period $t \in \{1,\ldots,T\}$ the set-up costs f_t and the production costs c_t are assumed to be nonnegative. Moreover, the demand d_t is assumed to be positive. Note that the last assumption also states that d_1 is positive. This implies that the first period is a production period. These assumptions do not influence the rigor of the analysis in the following, in the sense that the algorithms and proofs can easily be adapted to hold for instances of ELS where the assumptions do not hold.

In this section we present algorithms that calculate solutions of the LP-relaxation of UFL-ELS, shortly RUFL, and its dual. The time in which these solutions are calculated is $O(T\log T)$ in for general instances of ELS and $O(T)$ if the production costs are non-increasing.

$$\text{(RUFL)} \qquad \min \sum_{t=1}^{T} \left[f_t y_t + c_t(\sum_{\tau=t}^{T} x_{t\tau}) \right] \tag{2.3.6}$$

$$\text{s.t.} \quad \sum_{t=1}^{\tau} x_{t\tau} = d_\tau \qquad (1 \le \tau \le T) \tag{2.3.7}$$

$$x_{t\tau} \le d_\tau y_t \qquad (1 \le t \le \tau \le T) \tag{2.3.8}$$

$$x_{t\tau} \ge 0 \qquad (1 \le t \le \tau \le T) \tag{2.3.9}$$

$$y_t \ge 0 \qquad (1 \le t \le T) \tag{2.3.10}$$

The dual of RUFL can be stated as

$$\text{(DRUFL)} \qquad \max \sum_{\tau=1}^{T} d_\tau v_\tau \tag{2.3.11}$$

$$\text{s.t.} \qquad \sum_{\tau=1}^{T} d_\tau w_{t\tau} \leq f_t \qquad (1 \leq t \leq T) \tag{2.3.12}$$

$$v_\tau - w_{t\tau} \leq c_t \qquad (1 \leq t \leq \tau \leq T) \tag{2.3.13}$$

$$w_{t\tau} \geq 0 \qquad (1 \leq t \leq \tau \leq T) \tag{2.3.14}$$

In any feasible solution of DRUFL one can decrease $w_{t\tau}$ to $\max\{0, v_\tau - c_t\}$, as follows from 2.3.13. This does not affect feasibility in 2.3.12, and it has no influence on the objective function. Then DRUFL reduces to

$$\max \sum_{\tau=1}^{T} d_\tau v_\tau \tag{2.3.15}$$

$$\text{s.t.} \qquad \sum_{\tau=t}^{T} (d_\tau \bullet \max\{0, v_\tau - c_t\}) \leq f_t \qquad (1 \leq t \leq T) \tag{2.3.16}$$

We consider the formulation of DRUFL with the constraints (2.3.16). This program is highly structured, and can be solved by inspection fixing the v_τ in a forward manner. This has been shown by Krarup and Bilde [27]. The following algorithm handles variables in a backward fashion, i.e., in order of decreasing index. A consequence is that the variables need not be stationary. However, it results, quite trivially, in an $O(T\log T)$ algorithm. The concept of the backward algorithm for DRUFL will be given first. Contrary to the description in Wagelmans, van Hoesel and Kolen [52], the geometric interpretation will be emphasized to give a more intuitive idea of the method.

Initially, all the variables $v_1, \ldots, v_T$ are fixed at value zero. Then, one by one, the variables v_s are determined in a backward fashion. Thus, in the iteration where v_s is determined, the variables $v_{s+1}, \ldots, v_T$ have already been assigned values one or more times. Dual feasibility is not maintained during the algorithm. In fact, the solutions are partially dual feasible during the algorithm, i.e., at the beginning of the iteration where v_s is determined for the first time, the inequalities (2.3.16) are satisfied for the periods $s+1, \ldots, T$ and not necessarily for the other periods. The variables $v_{s+1}, \ldots, v_T$ are nonnegative and non-increasing.

The non-increasing order of the variables $v_{s+1},\ldots,v_T$ is formulated in the so-called *staircase property*. Periods $t_0,\ldots,t_P$ are identified, $T+1 \geq t_0 > t_1 > \ldots > t_P = s+1$, such that $v_{t_p} = v_{t_p+1} = \ldots = v_{t_{p-1}-1} > v_{t_{p-1}}$ for $p \in \{1,\ldots,P\}$. Finally $v_{t_0},\ldots,v_T = 0$. The periods $t_0,t_1,\ldots,t_P$ are called *potential* periods, since these are the potential successors of s as production periods in the algorithm. Note that the potential periods are indexed in DECREASING order. The staircase property is illustrated in the following picture, where on the horizontal axis the periods are drawn, and on the vertical axis the corresponding values of the dual variables are drawn.

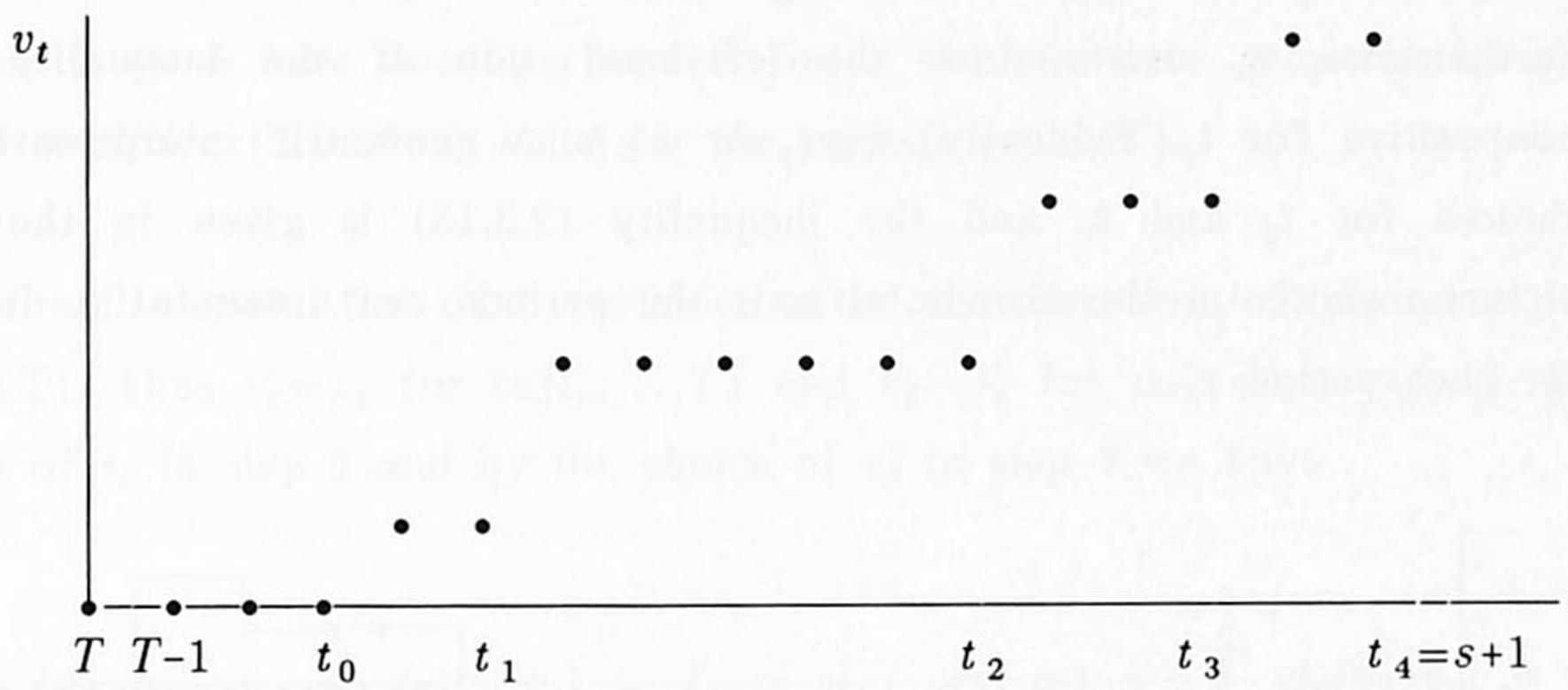


Figure 2.1. *The staircase property.*

A third property of the dual variables relates them to the value of the primal solution.

$$\sum_{\tau=t_p}^{T} d_\tau v_\tau = B(t_p) \qquad p \in \{0,\ldots,P\} \qquad (2.3.17)$$

Recall that $B(t)$ is the value of the optimal solution of the instance of ELS, restricted to the planning horizon consisting of the periods $t,\ldots,T$.

In the first iteration the dual variable v_T is set to $c_T + \frac{f_T}{d_T}$. It is easily seen that this solution satisfies the conditions: the staircase property, and 2.3.16 and 2.3.17 for period T.

Now suppose that $v_{s+1},\ldots,v_T$ are determined and that the conditions hold for the periods $s+1,\ldots,T$, where $t_0,\ldots,t_P$ are the potential periods. The iteration for period s consists of four steps. In the first two steps, two special potential periods t_q and t_r are identified.

STEP 1: t_q is chosen as the earliest potential period for which $v_{t_q} \leq c_s$.

STEP 2: t_r is chosen as the earliest potential period for which

$$d_{s,t_r-1}(v_{t_r}-c_s) + \sum_{\tau=t_r}^{t_q-1} d_\tau(v_\tau-c_s) \leq f_s \qquad (2.3.18)$$

Note that t_q exists by the nonnegativity of the production costs: $v_{t_0}=0\leq c_s$. Furthermore, t_r exists since the left-hand side of the inequality 2.3.18 is nonpositive for t_q. Evidently, $t_r \leq t_q$ or $r \geq q$. A geometric interpretation of the choices for t_q and t_r and the inequality (2.3.18) is given in the following picture, where on the horizontal axis the periods are drawn at a distance $d_{\tau T}$ for each period τ.

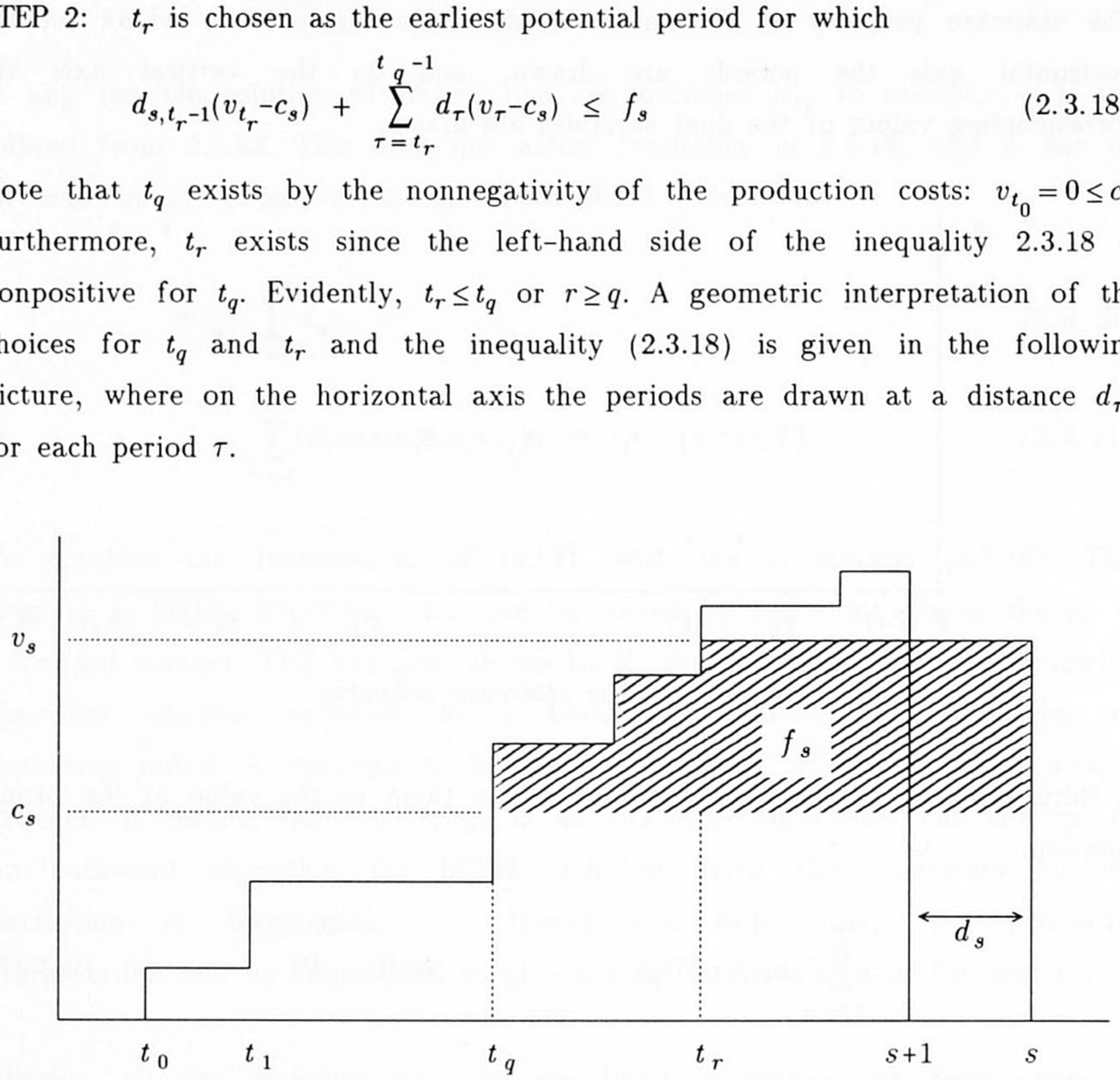


Figure 2.2. The determination of t_q and t_r.

In step 3 the dual variables are redetermined.

STEP 3: For $t=t_r,\ldots,T$, the variables v_t remain unchanged.
For $t=s+1,\ldots,t_r-1$, the variables v_t are set equal to v_s, where v_s is determined from

$$d_{s,t_r-1}(v_s-c_s) + \sum_{\tau=t_r}^{t_q-1} d_\tau(v_\tau-c_s) = f_s \qquad (2.3.19)$$

Geometrically, v_s is determined as drawn in figure 2.2.

STEP 4: The potential periods between s and t_r, i.e., the periods $t_{r+1}, t_{r+2}, \ldots, t_P$ are deleted from the set of potential periods. Period s becomes a potential period.

2.3.1. Lemma.

After steps 1 *to* 4 *have been performed for* s *the staircase property is still valid, and condition* (2.3.16) *is valid for the periods* $\{s,\ldots,T\}$.

Proof. We will denote the dual variables after the iteration of period s by v'_t, $(s \le t \le T)$. Thus $v'_t = v_t$ for $t \in \{t_r,\ldots,T\}$ and $v'_t = v'_s$ for $t \in \{s+1,\ldots,t_r-1\}$. By the choice of t_r in step 2 and by the choice of v'_s in step 3 we have

$$d_{s,t_r-1}(v_{t_r}-c_s) + \sum_{\tau=t_r}^{t_q-1} d_\tau(v_\tau-c_s) \le f_s = d_{s,t_r-1}(v'_s-c_s) + \sum_{\tau=t_r}^{t_q-1} d_\tau(v'_\tau-c_s)$$

It follows directly from this formula that $v_{t_r} \le v'_s$. This proves that the staircase property still holds, since $v'_t = v'_s$ for $t \in \{s,\ldots,t_r-1\}$, and $v'_t = v_t$ for $t \in \{t_r,\ldots,T\}$.

Since the variables $v_{t_r},\ldots,v_T$ have not changed, we need only prove $v'_t \le v_t$ for $t \in \{s+1,\ldots,t_r-1\}$. If $s+1 = t_r$, then this set is empty, and therefore we assume that $s+1 < t_r$. By the choice of t_r we have

$$d_{s,t_r-1}(v_{t_{r+1}}-c_s) + \sum_{\tau=t_r}^{t_q-1} d_\tau(v_\tau-c_s) \ge f_s = d_{s,t_r-1}(v'_s-c_s) + \sum_{\tau=t_r}^{t_q-1} d_\tau(v'_\tau-c_s)$$

Here, we used the fact that $v_\tau = v_{t_{r+1}}$ for $\tau \in \{t_{r+1},\ldots,t_r-1\}$. It follows immediately that $v'_{t_{r+1}} \le v_{t_{r+1}}$. Therefore, by the staircase property, no dual variables except v_s, of course, increase. This immediately implies that

(2.3.16) is satisfied for $t=s+1,\ldots,T$. The fact that s satisfies 2.3.16 follows immediately from the choice of v'_s.

□

The last one of the conditions, the equations 2.3.17 are left to be proved. After step 4 is executed, the potential periods are the periods $s,t_r,t_{r-1},\ldots,t_0$. For the potential periods $t_r,t_{r-1},\ldots,t_0$ the dual variables do not change, and therefore condition 2.3.17 is valid for them. It is left to prove that the following lemma holds for period s.

2.3.2. Lemma. $\sum_{\tau=s}^{T} d_\tau v_\tau = f_s + c_s d_{s,t_q-1} + B(t_q) = B(s)$

Proof. First, we prove $\sum_{\tau=s}^{T} d_\tau v_\tau \le B(s)$. Let UFL($s$) be UFL restricted to the periods $s,\ldots,T$. Then the dual of the LP-relaxation of UFL(s) becomes (after elimination of $w_{t\tau}$):

$$\text{DRUFL}(s) \qquad \max \sum_{\tau=s}^{T} d_\tau v_\tau \tag{2.3.20}$$

$$\text{s.t.} \qquad \sum_{\tau=s}^{T} d_\tau \max\{0, v_\tau - c_t\} \le f_t \qquad (s \le t \le T) \tag{2.3.21}$$

Since the constraints are exactly the constraints of DRUFL for $t=s,\ldots,T$ the solution $\{v'_\tau \mid s \le \tau \le T\}$ is feasible in DRUFL(s), and its value $\sum_{\tau=s}^{T} d_\tau v_\tau$ is a lower bound for the optimal value of UFL(s), which is $B(s)$ by definition..

Second, we prove that $f_s + c_s d_{s,t_q-1} + B(t_q) \ge B(s)$. This follows directly, since the left hand side is the cost of a feasible schedule, i.e., the schedule where production in s is equal to the demand of periods $s,\ldots,t_q-1$ combined with an optimal strategy from t_q to T, and $B(s)$ is the value of the optimal strategy.

Third, we prove that $\sum_{\tau=s}^{T} d_\tau v_\tau = f_s + c_s d_{s,t_q-1} + B(t_q)$. By rewriting $\sum_{\tau=s}^{T} d_\tau v_\tau$ we get

$$\sum_{\tau=s}^{T} d_\tau v_\tau = d_{s,t_r-1}(v_s-c_s) + \sum_{\tau=t_r}^{t_q-1} d_\tau(v_\tau-c_s) + \sum_{\tau=s}^{t_q-1} d_\tau c_s + \sum_{\tau=t_q}^{T} d_\tau v_\tau$$

From 2.3.19, it follows that $d_{s,t_r-1}(v_s-c_s) + \sum_{\tau=t_r}^{t_q-1} d_\tau(v_\tau-c_s) = f_s$. It is evident that $\sum_{\tau=s}^{t_q-1} d_\tau c_s = c_s d_{s,t_q-1}$. Finally, $\sum_{\tau=t_q}^{T} d_\tau v_\tau = B(t_q)$, since the condition 2.3.16 holds for t_q.

□

The importance of the lemma lies in the fact that it states that, if a schedule uses s as a production period, then the optimal strategy is to produce the demands for the periods $s,\ldots,t_q-1$ in s, and then follow an optimal strategy from t_q on with t_q as the succeeding production period of s.

Finally, note that the algorithm ends with a dual feasible solution after steps 1–4 are performed for $s=1$. Moreover, the third condition states that the value of an optimal production schedule is $\sum_{\tau=1}^{T} d_\tau v_\tau$ for the final dual solution.

The details regarding the implementation of the algorithm are specified for each step below:

STEP 1: The determination of t_q.

The values v_{t_p} $(p=0,\ldots,P)$ are increasing in p. Therefore, finding the highest variable with value smaller than or equal to c_s can be done using binary search. Note that $B(s) := f_s + c_s d_{t_q-1} + B(t_q)$ can already be determined, in this step.

STEP 2: The determination of t_r.

Rewriting 2.3.18 gives $d_{s,t_r-1}v_{t_r} + \sum_{\tau=t_r}^{t_q-1} d_\tau v_\tau \le f_s + c_s d_{t_q-1}$. Adding $\sum_{\tau=t_q}^{T} d_\tau v_\tau$ to the left and $B(t_q)$ to the right (they are equal) gives $d_{s,t_r-1}v_{t_r} + B(t_r) \le B(s)$. The left-hand side is monotone decreasing in r. This is easily seen after rewriting it to the following: $\sum_{\tau=s}^{t_r-1} d_\tau v_{t_r} + \sum_{\tau=t_r}^{T} d_\tau v_\tau = \sum_{\tau=s}^{T} d_\tau \bullet \min\{v_{t_r}, v_\tau\}$ and noticing that v_τ is monotone decreasing in τ. Therefore, we test for t_p $(p=P,\ldots,q)$ whether the above inequality holds or not. The first index encountered

for which the test holds is the desired t_r. If the test fails for a certain period, this period is deleted from the set of potential periods.

STEP 3: The determination of v_s.
Analogous to step 2 we can rewrite 2.3.19 as $d_{s,t_r-1}v_s = B(s)-B(t_r)$ from which v_s can be determined.

STEP 4: This step is implemented in the steps 2 and 3. A stack is used in which the potential periods and the information on the dual variables is maintained. Deletion of potential periods only takes place from the top, and addition of s is also done at the top of the stack.

In steps 2 and 3, cumulative demands $d_{t\tau}$ are used. Since the number of possible pairs (t,τ) is $\Omega(T^2)$, they should not all be calculated in advance, but only when necessary. This can be done in constant time using the formula $d_{t\tau} = d_{tT}-d_{\tau+1,T}$. Therefore, the values d_{tT} are calculated in advance, which takes a preprocessing of $O(T)$ time. The complexity of the algorithm is calculated for each step 1, 2 and 3 separately in the following lemma.

2.3.3. Lemma. *Step* 1 *takes* $O(logT)$ *per iteration*;
Step 2 *takes* $O(T)$ *overall*;
Step 3 *takes* $O(1)$ *per iteration.*

Proof. In step 1, a binary search is performed over P sorted values (the number of potential periods). Since $P \leq T$ this takes $O(\log\ T)$ time. In step 2, a test is performed for the earliest potential period left. If the test fails this period is deleted from the set of potential periods and will never return. Therefore, a test failure occurs at most T times (at most once per potential period) during the algorithm. If a test succeeds, then the index r has been found, and step 2 ends for the current iteration. Therefore, a succes also occurs at most T times (at most once per iteration). Concluding, step 2 takes $O(T)$ amortized time, i.e., the total time spent on this step during the complete algorithm is linear in T. In step 3, the value v_s is determined, which is easily seen to take constant time.

□

2.3.4. Theorem. *ELS can be solved in $O(TlogT)$ time.*

Proof. DRUFL is solved in $O(T\log T)$ time. Steps 2 and 3 of the dual algorithm take $O(T)$ time. The bottleneck is the binary search in step 1, which takes $O(T\log T)$ time, over T iterations. In order to find the optimal production plan after the dual is solved, it suffices to store t_q for each period s, which is the successor of s, if s is a production period (see lemma 2.3.2).

□

As one can observe in lemma 2.3.3, the bottleneck concerning the complexity of the algorithm is found in step 1. Logically, one might ask under which circumstances the binary search in this step can be replaced by a more efficient procedure. In fact, the answer follows directly from the fact that q is the maximal index p (note that the potential periods are decreasing in p) for which $v_{t_p} \leq c_s$. This index can be found without binary search, if the production costs c_t are ordered monotonically. In case the $\{c_t | t=1,\ldots,T\}$ are monotonically non-decreasing the problem is known to be solvable trivially by producing the total demand in period 1. So we consider the case in which the $\{c_t | t=1,\ldots,T\}$ are monotonically non-increasing.

Suppose that t_q is determined in step 1 of the iteration of s. Since $r>q$ the periods $t_0,\ldots,t_q$ remain in the potential set, which is $\{t_0,\ldots,t_r,s\}$. Therefore, since $c_{s-1} \geq c_s$, the period determined in step 1 in the following iteration is in the set $\{t_q,\ldots,t_r,s\}$. To find it, a linear search is done instead of a binary search on this set, where a test $v_t > c_{s-1}$ is performed for the periods $t_q,\ldots t_r,s$. Any time the test fails we move a period upward in the set and we do not consider the period t in the test in the future. Therefore, we have $O(T)$ failures during the whole algorithm. If the test succeeds we stop. Then we have found the earliest potential period for which the inequality is invalid. Note that if the test also fails for s, then this is the period we are looking for. The number of successes is therefore bounded by T (the number of iterations). We may conclude that step 1 can now be performed in $O(T)$ amortized time. This leads to the following theorem. Recall that $c_t = p_t + h_{tT}$.

2.3.5. Theorem.
Instances of ELS, where c_t $(t=1,\ldots,T)$ *are non-increasing or equivalently* $p_t+h_t \geq p_{t+1}$ $(t=1,\ldots,T-1)$ *are solvable in* $O(T)$ *time.*

The case considered by Wagner and Whitin consists of stationary production costs $p_t=p$ $(t=1,\ldots,T)$ and nonnegative holding costs h_t. The condition in theorem 2.3.5. is therefore fullfilled for this case.

A corollary of the algorithm is the following result on the polyhedral description of the UFL-formulation of ELS

2.3.6. Theorem.
The LP-relaxation of UFL-ELS always yields integral optimal solutions for the cost functions occurring in ELS.

2.4. Concluding remarks.

This chapter contains two types of results. The most important results concern the improved complexity for algorithms to solve ELS. This complexity is derived by a linear programming based algorithm. However, there is a much more natural formulation for ELS in the form of a dynamic program. Since the linear programming formulation from section 2.3, is closely related to the dynamic program, as can already be concluded from the algorithm itself, it should not be surprising that there is also an algorithm which solves the (backward) dynamic programming recursion for ELS with the same complexity. This is the topic of the following chapter. Moreover, it turns out that there is also an algorithm that solves the forward dynamic programming recursion in the same time complexity. Therefore, the (forward) algorithm of Krarup and Bilde [27] can also be implemented such that it runs in the same time bounds as derived in section 2.3.

Chapter 3

IMPROVED DYNAMIC PROGRAMMING ALGORITHMS FOR THE ECONOMIC LOT SIZING PROBLEM AND EXTENSIONS

3.1. Introduction

As argued by Denardo [8], the complexity of a dynamic programming based algorithm is usually proportional to the number of arcs in the underlying dynamic programming network. Only when special cost structures on the arcs of the network are incurred one might hope for a better running time. One of the first improvements of such nature is due to Yao [58], who uses a monotonicity property to speed up the dynamic programming algorithm for finding optimal binary search trees. A similar monotonicity property holds for the dynamic programming formulation of ELS. Moreover, this property can be interpreted here in geometrical terms. Consequently, implementations of both the forward and backward dynamic programming recursions result which are very elegant, and have a better running time than the $O(T^2)$ method of Wagner and Whitin. This running time is $O(T\log T)$ for general cost structures, and $O(T)$ in special cases of ELS. However, there is a striking difference between the implementations of the forward and the backward recursions, with respect to the necessary data structures. To explain this, and the better running time for the special case of ELS, the bare geometric techniques are explained in some detail.

There are two extension of ELS which can be solved by dynamic programming recursions similar to the one for ELS. The first extension allows for backlogging. It is denoted by ELSB. Zangwill [60] introduced this model. He considered the more general case in which all cost functions are concave, and he showed that under this assumption a generalization of the zero-inventory property for ELS holds for ELSB. This property makes it possible to use a

dynamic programming approach to solve the model in $O(T^3)$ time. Later, in Zangwill [61], he considered the special case in which the production and holding costs are of linear type with fixed charges, and he proposed an $O(T^2)$ dynamic programming algorithm based on a double recursion, to solve this special case of ELSB. In this chapter, we show how the the geometric techniques can be used to solve the double recursion in $O(T\log T)$ time for general linear costs, and we give conditions under which a linear time implementation is possible. The second extension of ELS incurs start-up costs besides the set-up costs. It is denoted by ELSS. Although this type of fixed cost structure has been introduced by Schrage [48] the model for ELSS has first been studied by Karmarkar and Schrage [23], Karmarkar, Kekre and Kekre [24], and Van Wassenhove and Vanderhenst [55]. Since this model only has a different fixed cost structure compared to the cost structure of ELS, the same zero-inventory property holds. Therefore, the dynamic programming recursion for ELSS is adapted from that of ELS only to account for this fixed cost structure. This also results in a double recursion which can be solved in the same time bounds as given above, i.e., in $O(T\log T)$ time in general, and in $O(T)$ time for special cost structures.

The method that will be developed in this chapter can be implemented fairly easily. Since the worst-case running time of this method is better than the worst-case running time of existing implementations, like Evans [13], and Saydam and McKnew [47] among others, it is interesting to compare these methods when run on several types of instances of ELS. Finally, it should be mentioned that algorithms with the same time complexity have been developed by Aggarwal and Park [1], who consider ELS and ELSB (and some other lot sizing problems). Their method will also be part of the computational comparisons in this chapter.

In section 3.2, the geometric techniques used in the implementations of the algorithms for the dynamic programming recursions of ELS and its extensions, are discussed. In section 3.3, it is shown how the dynamic programming recursion of the economic lot sizing problem is formulated to fit in the description necessary to apply the geometric techniques. Also, the planning horizon theorems for ELS are proved with these geometric techniques. In

section 3.4, the extensions with backlogging and start-up costs are evaluated. Finally, in section 3.5, computational experiments are described.

3.2. Geometric solution techniques for dynamic programs

A basic topic in computational geometry is the determination of the lower concave envelope of a given set of lines in the plane. A dual problem, in a sense, is the problem of determining the convex hull of a given set of points in the plane. For both problems efficient solution techniques exist. These solution techniques will be used to solve the following type of recursion formulae for dynamic programs.

$$\begin{aligned} G(0) &:= 0; \\ G(i) &:= A_i + \min_{0 \le j < i} \{G(j) + B_j + C_i D_j\} \qquad (1 \le i \le n) \end{aligned} \tag{3.2.1}$$

Here A_i, B_j, C_i, D_j $(1 \le i, j \le n)$ are constants that only depend on the index. In this section, these constants are supposed to be given beforehand.

We proceed by showing how the dynamic programming recursion 3.2.1 can be viewed as a problem of finding the lower concave envelope of a set of lines. To calculate the minimum of the terms $\{G(j)+B_j+C_iD_j | 0 \le j < i\}$ for a fixed i, these terms are considered as functions of C_i. Each term corresponds to a line or a linear function m_j $(0 \le j < i)$, where $m_j(x) = G(j)+B_j+D_jx$,i.e., m_j is the line with slope D_j that passes through the point $(0, G(j)+B_j)$. Obviously, the determination of the minimum of $\{G(j)+B_j+C_iD_j | 0 \le j < i\}$ is equivalent to finding the minimum of the values $m_j(C_i)$ over j: $0 \le j < i$. This minimum is defined as $g_i(C_i)$, where g_i is the concave lower envelope of the lines m_j $(0 \le j < i)$. Note that g_i is piecewise linear. Given $g_i(C_i)$, one can determine the dynamic programming variable $G(i)$, which is simply $A_i + g_i(C_i)$. An algorithm based on this observation proceeds by calculating the variables $G(i)$ $(i = 1, \ldots, n)$, iteratively, in a forward fashion. During this process the lower envelope of the lines is updated, since a new line m_i is added to the set of lines in iteration i. The operations in each iteration are divided in the following two steps:

STEP 1: Find the value $g_i(C_i)$, the function value of the lower envelope of the lines m_j $(j<i)$ at C_i. With this value $G(i)$ is calculated by adding A_i to $g_i(C_i)$.

STEP 2: Add the line m_i to the set of lines that already determine the lower concave envelope g_i, where m_i is defined as $m_i(x)=G(i)+B_i+D_ix$. Recalculate the lower envelope g_{i+1}, where $g_{i+1}(x) := min\{g_i(x), m_i(x)\}$.

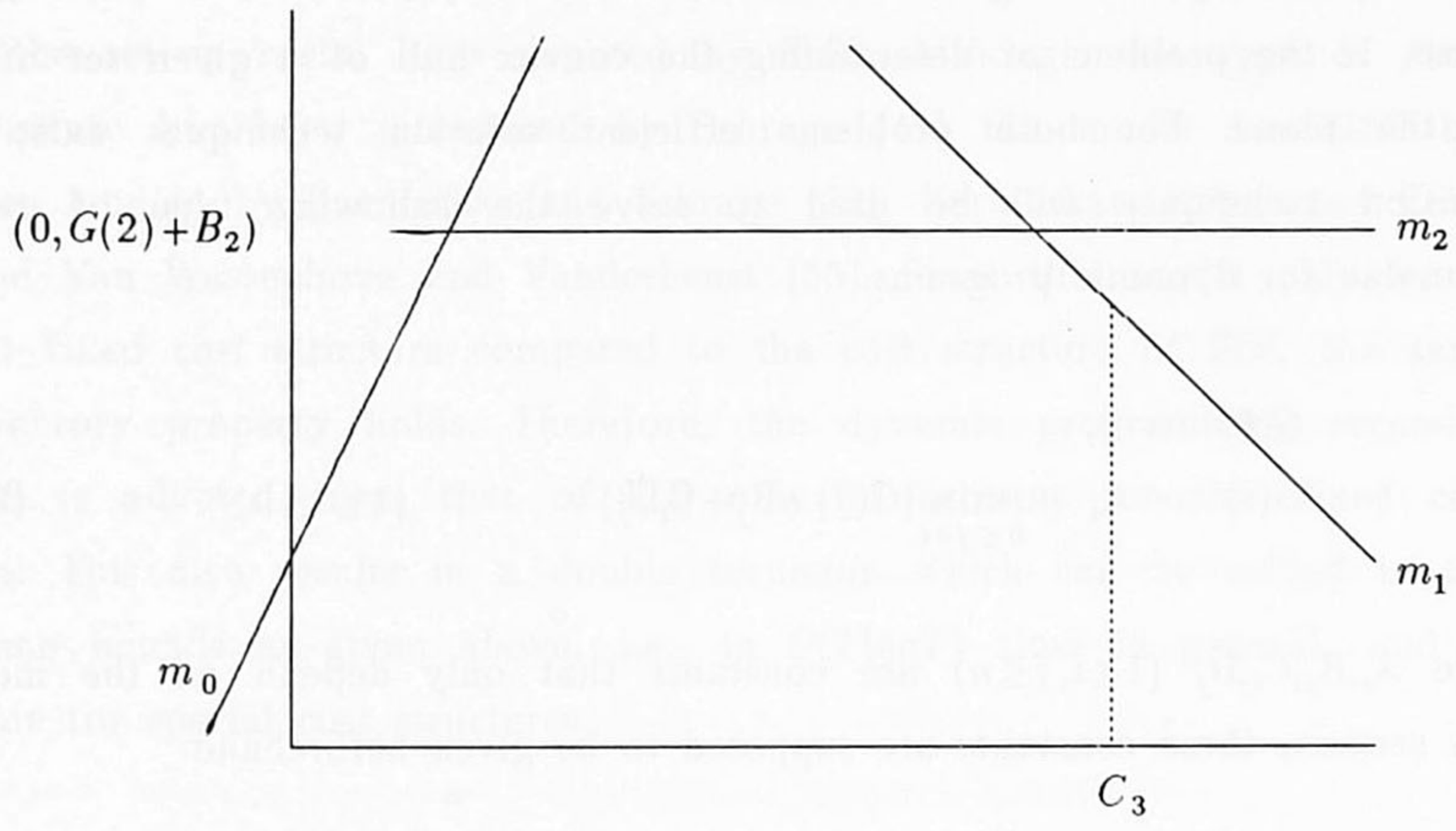


Figure 3.1. *The lower concave envelope of a set of lines.*

For an analysis of the running time of an algorithm based on this description a suitable datastructure to maintain the lower envelope g_i should be chosen. In order to choose the right structure it is necessary to find out which elementary operations must be performed. Therefore, the two steps of the algorithm will be analyzed in some more detail.

In step 1 of the iteration where $G(i)$ is determined, the function value of g_i at $x=C_i$ has to be calculated. This amounts to the search of the line segment of g_i that contains C_i.

In step 2 the line m_i is added to the concave lower envelope. In order to do so, the points where m_i and g_i meet should be calculated, if they exist, and the line segments of g_i between these points, should be replaced by m_i. This procedure is separated in the following two steps.

STEP 2A: First, detect a breakpoint of g_i that lies above m_i, if one exists. Since the line segments of g_i have decreasing slopes, when viewed in order of increasing x, it suffices to search the breakpoint B, for which the left line segment incident to B, has slope larger than D_i and the right line segment has slope at most D_i. If this breakpoint lies above the line m_i, then the lower envelope should be updated. If no such breakpoint exists, then it follows from the concavity of g_i that $g_{i+1}=g_i$ and therefore, we are finished with step 2 in that case.

Note that there are two exceptions, namely the case in which all line segments have larger slopes, and the case in which all line segments have smaller slopes. In the first case it suffices to check whether the right-most breakpoint of g_i lies above m_i and/or whether the right-most line segment meets m_i. In the second case it suffices to check whether the left-most breakpoint of g_i lies above m_i and/or whether the left-most line segment meets m_i.

STEP 2B: Now suppose that the breakpoint P of g_i lies above m_i. Then first the line segments to the right of P are inspected. Let the breakpoint of the closest segment right of P be denoted by Q. There are two possibilities: Q lies below m_i or Q lies above (or on) m_i. In the first case m_i meets the line segment defined by P and Q. The meeting point is calculated, say it is R, and the line segment $[P,Q]$ is deleted. Moreover, the line segment $[R,Q]$ is added. In the second case the line segment $[P,Q]$ is deleted and we proceed by inspecting the line segment right of Q in the same manner, until a meeting point R is found. An analogous procedure inspects the line segments left of P, until a meeting point L is found. Finally, the line segment $[L,R]$ is added to g_{i+1}.

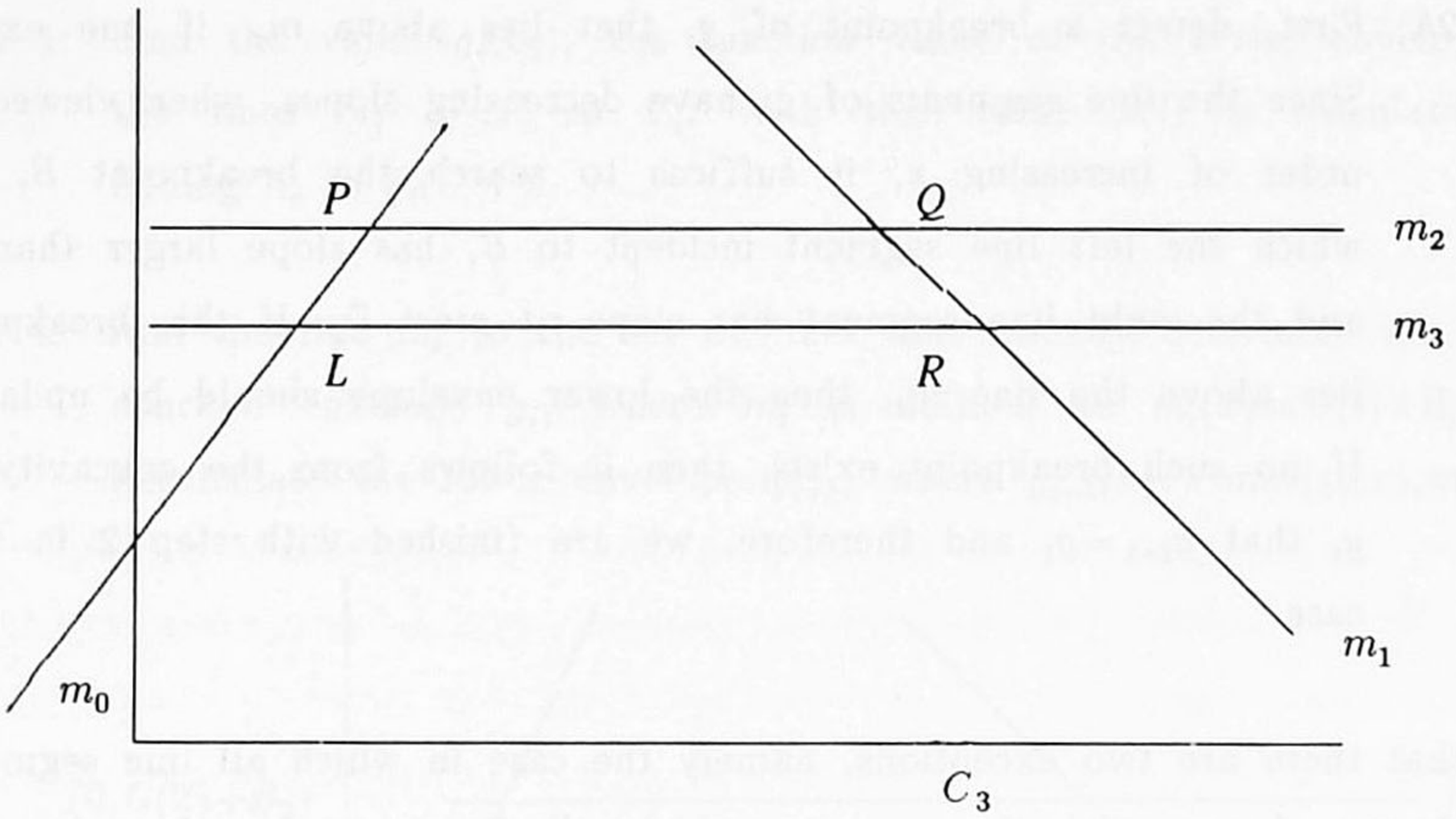
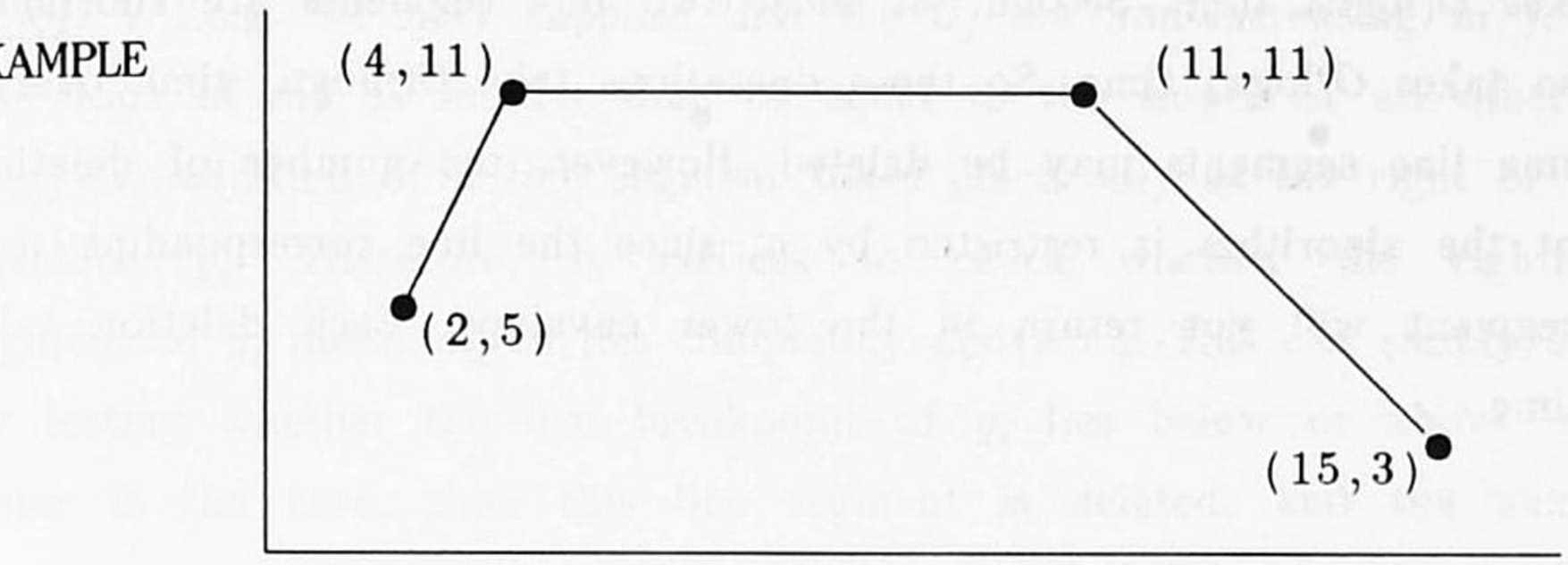


Figure 3.2. *Calculation of* $g_4(x)$.

A suitable datastructure that maintains the lower envelope should be able to perform the following operations quickly. In step 1 a line segment must be determined which contains the point with a given value of x. In step 2A a breakpoint should be determined where the slopes of the line segments meeting this point, s_l and s_r, satisfy $s_l \leq s < s_r$ for a given number s. In step 2B a neighboring line segment should be detected. Moreover, one or more line segments may be deleted and possibly a line segment may be added.

Thus the operations that have to be performed are FIND, ADD and DELETE. A balanced tree, like a 2-3 tree (or an AVL-tree) supports these operations in time proportional to the logarithm of the number of nodes, see Aho et al. [2]. The 2-3 tree is designed such that the leaves contain the line segments of the lower envelope. The information maintained in each leaf consists of the endpoints of the line segment including their value, and the slope of the line segment. The number of line segments is bounded by n, since each line segment corresponds to exactly one line m_j. The intermediate nodes of the tree contain the interval that is covered by the line segments in the subtree of the intermediate node. Moreover such a node contains information on the leftmost breakpoint and the leftmost (highest) slope, and it contains the same information on the rightmost breakpoint and slope in the subtree.



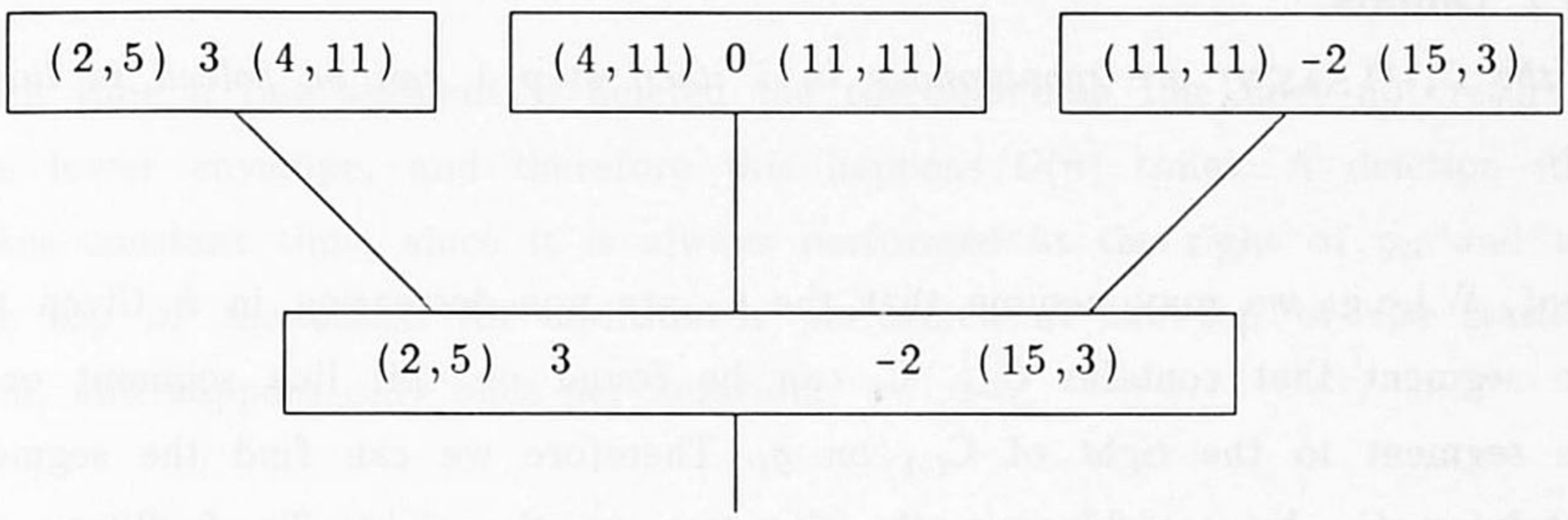


Figure 3.3. *Partial lower envelope and corresponding nodes in* 2–3 *tree.*

3.2.1. Theorem.

The dynamic programming recursion 3.2.1 *can be solved in* $O(n\log n)$ *time.*

Proof. Due to the choice of the data structure the basic operation of determining the value $g_i(x)$ for some x, takes $O(\log n)$ time. Moreover, the deletion and addition of a line segment also takes $O(\log n)$ time. It remains to be shown, therefore, that the number of times that these operations are executed is bounded by $O(n)$.

Step 1 takes $O(\log n)$ time per iteration, since only $g_i(C_i)$ must be determined, and there are n iterations.

Step 2A takes $O(\log n)$ time per iteration, since only a breakpoint for which the slopes of the left and right line segments are larger and smaller than D_i, respectively.

Step 2B contains several operations. First, at most one line segment is added,

which takes $O(\log n)$ time. Second, at most two line segments are shortened, which also takes $O(\log n)$ time. So these operations take $O(n\log n)$ time overall. Third, some line segments may be deleted. However, the number of deletions throughout the algorithm is restricted by n, since the line corresponding to a deleted segment will not return in the lower envelope. Each deletion takes $O(\log n)$ time.

□

In the following we show under which conditions the different steps can be implemented in linear time.

3.2.2. Lemma.
If the C_i $(1 \le i \le n)$ are monotonous in i, then step 1 can be solved in linear time.

Proof. W.l.o.g. we may assume that the C_i are non-decreasing in i. Given the line segment that contains C_{i-1}, C_i can be found on this line segment or a line segment to the right of C_{i-1} on g_i. Therefore we can find the segment containing C_i, by searching in the direction to the right. To facilitate the location of a neighboring right leaf in the 2-3 tree, a doubly linked list is introduced, which connects neighboring leaves. Each time a segment is inspected, either we pass it (if C_i is not contained in it) or we stop, (if C_i is contained in it). A stop happens only once per iteration, and therefore there are $O(n)$ stops. If a pass happens, then the line segment will not be inspected anymore, since the C_i are non-decreasing. Therefore, the line corresponding to the segment is not considered in the rest of the algorithm, anymore. Because the number of lines is $O(n)$, passes also happen only $O(n)$ times. The doubly linked list is incorporated to find a neighbor segment in constant time.

□

3.2.3. Lemma.
If the D_j $(1 \le j \le n)$ are monotonous in j, then STEP 2 can be performed in linear time. Moreover, the $2-3$ tree can be replaced by a stack.

Proof. W.l.o.g. we may suppose that the D_j are non-increasing in j. Since D_i, the slope of m_i, is smaller than or equal to the slopes of all other lines m_j $(0 \leq j < i)$ addition of a line segment takes place only at the right of the lower envelope g_i. Therefore, it suffices to check whether the rightmost line segment of g_i meets m_i or lies completely above m_i. This can simply be checked by testing whether the last breakpoint of g_i lies below or above m_i. If the latter is the case, then this line segment is deleted, and the next to last line segment will be inspected, and so on, until a line segment is found that meets m_i. Then the intersection point of m_i and that line segment is calculated, and this point together with its corresponding line segment are added to the lower envelope.

Each time a line segment is deleted the corresponding line does not return in the lower envelope, and therefore this happens $O(n)$ times. A deletion itself takes constant time, since it is always performed at the right of g_i, and thus the top of the stack. An addition is performed at the top of the stack as well, and happens only once per iteration.

Finally, it should be noted that the stack can also be used to perform binary searches in order to find the value of C_i in the lower envelope.

□

Combining the lemmas 3.2.2 and 3.2.3. we get:

3.2.4. Corollary.

If the C_i and D_j $(1 \leq i, j \leq n)$ are monotonous with respect to their indices, the dynamic programming recursion 3.2.1 *can be solved in time proportional to n by using a stack as the datastructure to maintain the lower envelope.*

Proof. From lemma 3.2.3, it follows that a stack can be used to implement step 2 in $O(n)$ time. Since step 1 can be implemented in linear time using a doubly linked list, it can also be implemented with a stack, since the search features of a list are trivially implemented in a stack.

□

A final remark regards the constants A_i, B_j, C_i and D_j $(1 \leq i, j \leq n)$. These constants need not be available at any time in the procedure, but it suffices that they can be evaluated in constant time at the moment they should be available.

Another geometric technique, one that is closely related to the concept of using the lower concave envelope of lines, considers the lower convex envelope of a set of points. It is used to solve the following variant of the dynamic programming recursion 3.2.1.

$$G(0) := 0;$$
$$G(i) := a_i + \min_{j<i}\{G(j)+b_j+c_i(d_i-d_j)\} \qquad (1 \leq i \leq n) \qquad (3.2.2)$$

At the end of this section we will give the relation between both dynamic programming recursions. Moreover, it will be shown that both geometric concepts are dual to each other. Because of this close relation, it will not come as a surprise that the results concerning the complexity of the algorithm to solve 3.2.2 resemble those already discussed in this section, concerning 3.2.1.

Again, we start by describing what happens in an iteration of the algorithm. Suppose that for each j smaller than i, the corresponding variable $G(j)$ has been determined. Consider the points $(d_j, G(j)+b_j)$ in the plane. Through each of the points a line with slope c_i is drawn. For a fixed j, this line, denoted by l_j, is defined as the following function of x: $l_j(x) = G(j)+b_j+(x-d_j)c_i$. Since $l_j(d_i)$ is exactly the term between braces for a fixed j, we are interested in $\min\{l_j(d_i) | 0 \leq j < i\}$ and therefore the line l_j $(j<i)$ lying lowest at $x = d_i$ is the line to be found. This line is easily found, if the convex lower envelope g_i of the set of points $\{(d_j, G(j)+b_j) | j<i\}$ is available. Finding the lowest lying line, then amounts to finding the breakpoint of this envelope, where the line segments adjacent to this point change slope from smaller than c_i to bigger than (or equal to) c_i.

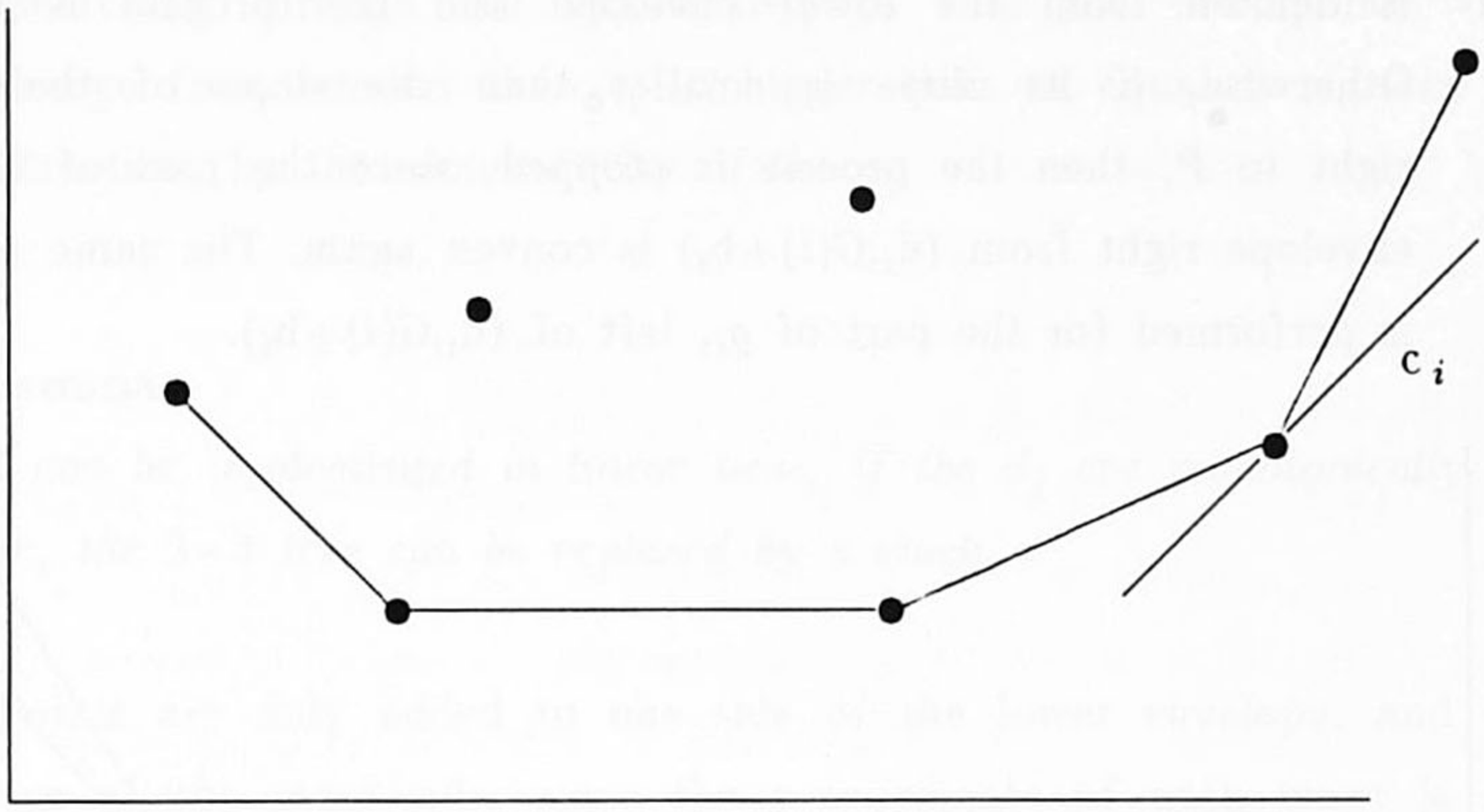


Figure 3.4. *The lower convex envelope with a tangent with slope* c_i.

The lower envelope is maintained by a 2-3 tree in which the leaves contain the line segments, with the endpoints, with function value, and the slopes, in order of increasing x-coordinate. The intermediate nodes contain all information of the first leaf contained in their subtree, together with the length of the interval that is covered by the line segments in the subtree. In each iteration, two steps have to be performed, one to determine $G(i)$ and one to update the lower envelope.

STEP 1: Search for the breakpoint, where the line segments adjacent to it, change slope from smaller than c_i to bigger than c_i.

STEP 2: Add $(d_i, G(i)+b_i)$ to the lower envelope, and update it.

STEP 2A: Determine the position of this point, i.e., search for the line segment of g_i that contains d_i. If it turns out that $(d_i, G(i)+b_i)$ lies above g_i, then we stop. Otherwise step 2B is performed, the reconstruction of the lower envelope.

STEP 2B: The line segment that contains d_i is checked in both endpoints. First, the endpoint to the right of d_i, say P, is checked. If the slope of the line segment between $(d_i, G(i)+b_i)$ and P is bigger than the slope of the line segment to the right of P, this line segment

is deleted from the lower envelope and this process is repeated. Otherwise, if its slope is smaller than the slope of the segment right to P, then the process is stopped, since the part of the lower envelope right from $(d_i, G(i)+b_i)$ is convex again. The same procedure is performed for the part of g_i, left of $(d_i, G(i)+b_i)$.

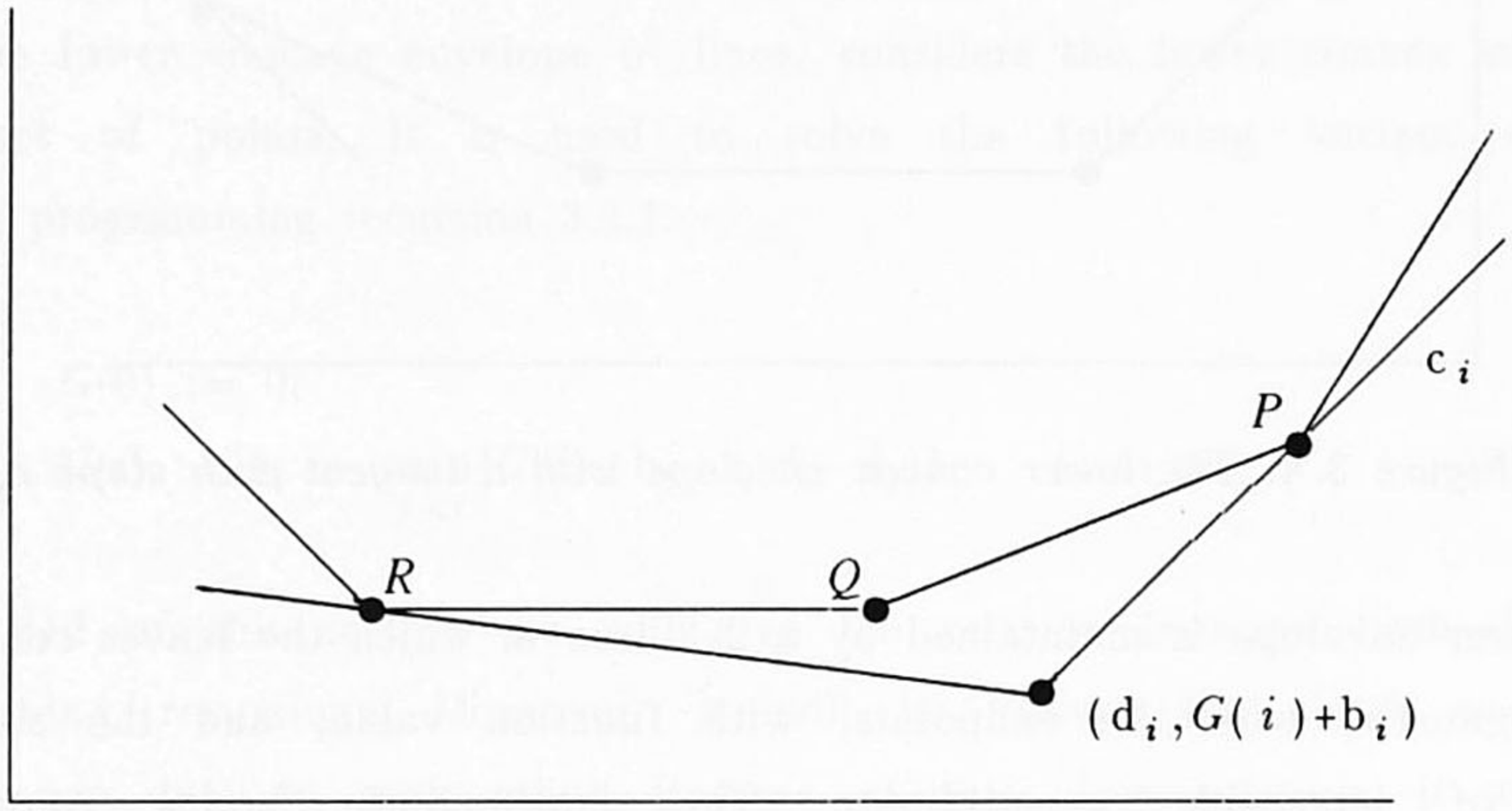


Figure 3.5. *The addition of a point to the lower convex envelope.*

The operations that take place in the steps 1 and 2 are easily seen to take $O(\log n)$ time in a 2–3 tree, since they amount to searching, addition and deletion of line segments. Moreover, analogous to the reasoning with respect to the previous geometric algorithm in this section, it is fairly easy to see that the operations are performed $O(n)$ times. This provides the proof for the following theorem.

3.2.5. Theorem.

The geometric implementation of the dynamic programming recursion 3.2.2. *takes* $O(n\log n)$ *time.*

Analogous to the implementation of 3.2.1, we can speed up the algorithm if the c_i and/or d_i are monotonically ordered with respect to the index i.

3.2.6. Lemma.

STEP 1 *can be implemented in linear time, if the* c_i *are ordered, by using a doubly linked list over the leaves of the* 2–3 *tree.*

Proof. Searching through the linked list, to find the breakpoint where the slopes change from smaller than c_i to bigger than or equal to c_i, can be done in one direction, by the monotonicity of the c_i.

□

3.2.7. Lemma.

STEP 2 can be implemented in linear time, if the d_j are monotonically ordered. Moreover, the 2–3 tree can be replaced by a stack.

Proof. Points are only added to one side of the lower envelope, and deletions take place at the same side, since the x-coordinate of each point is d_j. So a stack can be used to store the information. Moreover, since operations are only performed on the top of the stack no search is necessary, which implies that the running time is bounded by the number of deletions plus additions to the stack. It has already been argued that this number is bounded by $O(n)$.

□

3.2.8. Theorem.

If the c_i and d_j are monotonous, then the dynamic programming recursion 3.2.2 *is solvable in linear time.*

Proof This follows directly from the previous two lemmas.

□

Finally, the relation between the dynamic programming recursions, and the relation between the geometric techniques are discussed. The dynamic programming recursions are easily seen to be equivalent, by the following substitutions, performed on 3.2.2: $A_i=a_i+c_id_i$; $B_i=b_i$; $C_i=-c_i$ and $D_i=d_i$ $(i=1,\ldots,n)$.

The geometric techniques are dual in the sense that the roles the points and lines play in the first technique are exchanged in the second one. In the first technique each index corresponds to a line, and in the second technique each index corresponds to a point. Moreover, the lower envelope of lines corresponds to the lower envelope of points. More about the duality of both concepts can be found in Preparata and Shamos [40].

REMARK: The results of the theorems 3.2.4 and 3.2.8, can be further generalized as has been done by Klawe [26], among others. In this paper the following dynamic programming recursion is solved:

$$G(i) = \min_{j>i}\{G(j) + w_{ij}\} \qquad (3.2.3)$$

where the w_{ij} $(1 \le i < j \le n)$ satisfy a monotonicity condition. This condition states that for each i and j $(i<j)$: $w_{i,j} + w_{i+1,j+1} \ge w_{i+1,j} + w_{i,j+1}$. In fact, it suffices to demand a monotonicity with respect to the column minima in the matrix of elements w_{ij}. Note that if the c_i and d_j are monotone, then the dynamic programming recursions 3.2.1 and 3.2.2 satisfy the monotonicity condition: take $w_{ij} := A_i + B_j + C_i D_j$. The solution method of Klawe can therefore be used to solve the linear case of the dynamic programming recursion. However, a Divide and Conquer strategy, applied when the C_i and D_j are not monotone, provides an $O(n\log n)$ algorithm for the general case as well. This strategy is described in Aggarwal and Park [1].

3.3. Improved dynamic programming algorithms for ELS

The forward recursion and the backward recursion for ELS fit into the descriptions of the dynamic programming recursions formulated in section 3.2. Surprisingly, there are some conceptual differences between the two recursions. To show these differences we show how both recursions can be viewed as special cases of the dynamic programs 3.2.1 and 3.2.2. A brief description of the algorithms is also included in this section. Moreover, the similarities between the backward algorithm and the linear programming algorithm of section 2.3 will be pointed out. Finally, it is shown how some planning horizon theorems can be proved with the geometric interpretation of both algorithms.

Forward recursion

The forward recursion can be rewritten as

$$F(0):=0; \quad F(t) = \min_{0<\tau\le t}\{F(\tau-1)+(f_\tau - c_\tau d_{1,\tau-1})+c_\tau d_{1t}\} \ (t=1,\ldots,T) \qquad (3.3.1)$$

Recall the recursion 3.2.1 which is $G(0):=0; G(i)=A_i+\min_{j\le i}\{G(j)+B_j+C_iD_j\}$ $(1\le i\le n)$. It follows that t takes the role of i in 3.2.1 and that τ takes the role of j. Moreover, A_i is replaced by zero, B_j is replaced by $f_\tau - c_\tau d_{1,\tau-1}$, C_i is replaced by d_{1t} and D_j is replaced by c_τ. Especially the last two replacements turn out to be important because d_{1t} is monotone in t.

An iteration of the forward algorithm is described as follows. Suppose that the $F(\tau-1)$ $(\tau\le t)$ are determined, and consider the points $(d_{1,\tau-1}, F(\tau-1)+f_\tau)$ for $\tau=1,\ldots,t$. The line m_τ is defined as the line through this point with slope c_τ, i.e, m_τ is the line $m_\tau(x)=F(\tau-1)+f_\tau+c_\tau(x-d_{1,\tau-1})$. Now the value of m_τ in $x=d_{1t}$ is $F(\tau-1)+f_\tau+c_\tau d_{\tau t}$. Therefore, $F(t)$ is determined as $\min_{\tau\le t}\{m_\tau(d_{1t})\}$.

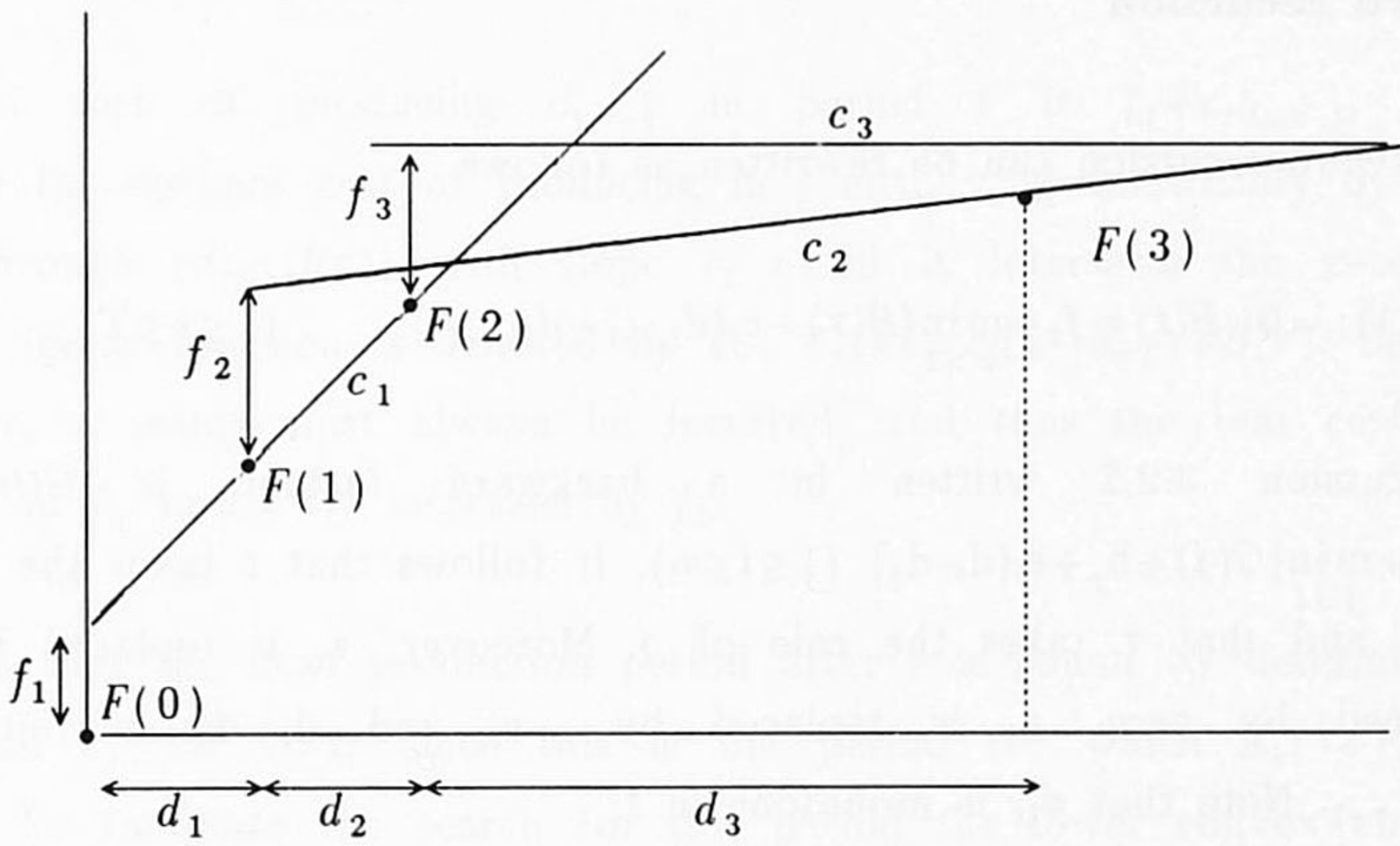


Figure 3.6. *The lower envelope for the forward recursion.*

Once $F(t)$ is determined, the line $m_{t+1}(x)=F(t)+f_{t+1}+c_{t+1}(x-d_{1t})$ is added to the set of lines $\{m_\tau|\tau\le t\}$. The two steps that have to be performed to accomplish this are the following.

STEP 1: Evaluate the value of the lower concave envelope g of the lines m_τ $(1\le\tau\le t)$ at $x=d_{1t}$. This value is $F(t)$.

STEP 2: Add the line m_{t+1} to g, where m_{t+1} passes through $(d_{1t}, f_{t+1}+F(t))$ and its slope is c_t.

Since d_{1t} are monotone non-decreasing in t, step 1 takes constant time in each operation, as has been proved in lemma 3.2.6. However, since the c_τ need not be monotone, a 2-3 tree is necessary to maintain the lower envelope. Therefore, step 2 is the bottleneck, since this step takes $O(T\log T)$ time during the algorithm. In case the c_τ are monotone, the operations in step 2 can be performed in linear time, as has been proved in lemma 3.2.7. Thus, the following theorem is immediate.

3.3.1. Theorem.

The forward recursion for ELS can be solved in $O(T\log T)$ time, using a $2-3$ tree to store the lower envelope of lines. If the production costs c_τ are monotone (non-increasing), then the forward recursion can be solved in $O(T)$ time.

Backward recursion

The backward recursion can be rewritten as follows.

$$B(T+1):=0;\ B(t)=f_t+\min_{\tau>t}\{B(\tau)-c_t(d_{1,t-1}-d_{1,\tau-1})\} \qquad (1\le t\le T) \qquad (3.3.2)$$

The recursion 3.2.2 written in a backward fashion is $G(n+1):=0$; $G(i)=a_i+\min_{j>i}\{G(j)+b_j+c_i(d_i-d_j\}$ $(1\le i\le n)$. It follows that t takes the role of i in 3.2.2, and that τ takes the role of j. Moreover, a_i is replaced by f_t, b_j is replaced by zero, c_i is replaced by $-c_t$ and d_i-d_j is replaced by $d_{1,t-1}-d_{1,\tau-1}$. Note that d_{1t} is monotone in t.

We describe an iteration in which $B(t)$ is determined, and thus $B(\tau)$ for $\tau>t$ are supposed to be known. For $\tau>t$ the points $(d_{\tau T},B(\tau))$ are plotted in the plane. (Note that $(d_{T+1,T},B(T)):=(0,0)$), So cumulative demand is put on the horizontal axis (the x-axis), and the vertical axis (the y-axis) corresponds to the minimal costs.

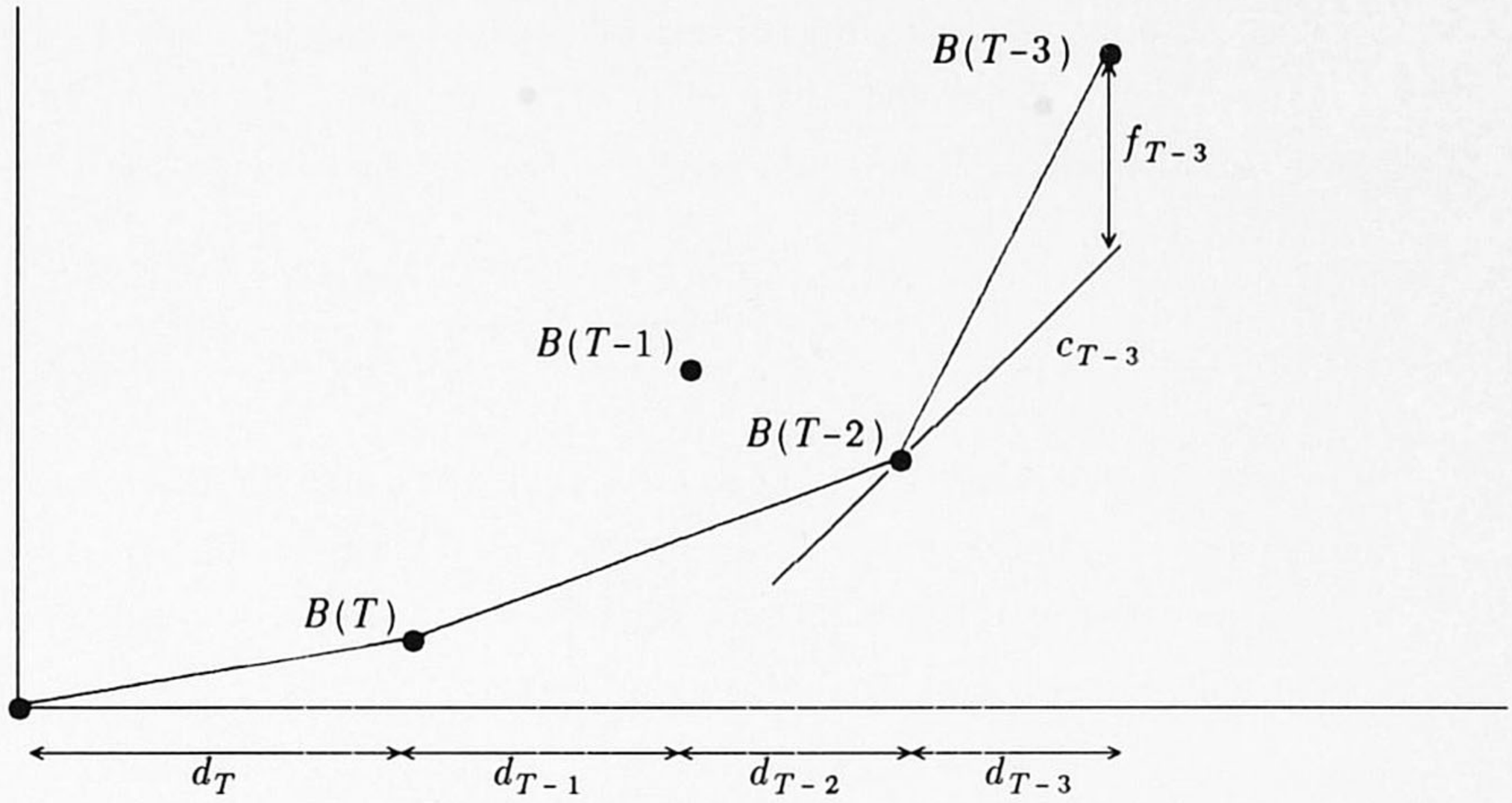


Figure 3.7. *The lower envelope for the backward recursion.*

Since the cost of producing $d_{t,\tau-1}$ in period t is $f_t+c_t d_{t,\tau-1}$, we can determine the optimal cost of producing in t until τ, geometrically by drawing a line through $(d_{\tau T}, B(\tau))$ with slope c_t untill it intersects the x-coordinate d_{tT}. This linear function is denoted by ℓ_τ: $\ell_\tau(x)=c_t(x-d_{\tau T})+B(\tau)$. Because d_t is positive, a setup must always be incurred, and thus the real cost $B(t)$ is the value of ℓ_τ in $x=d_{tT}$ increased by f_t.

It follows that the next production period after t is found by determining the lowest line ℓ_τ for $\tau>t$, since this is the period for which $B(\tau)+c_t d_{t,\tau-1}$ is minimal. To facilitate the search for this period the lower convex envelope g of the points $\{(d_{\tau T}, B(\tau))\,|\,\tau>t\}$ is constructed (see figure 3.7). The breakpoints of g correspond to points $(d_{\tau T}, B(\tau))$ for $\tau\in\{t+1,\ldots,T+1\}$. We denote the periods corresponding to the breakpoints by $T+1=t_0>t_1>t_2>,\ldots,>t_P=t+1$. The slopes of the line segments between t_p and t_{p+1} are denoted by v_{t_p} $(p=1,\ldots,P)$, are in increasing order of p, because of the convexity of g.

STEP 1: The period τ for which $B(\tau)+c_t d_{t,\tau-1}$ is minimal is denoted by t_q. It is the period for which the slopes left to $(d_{t_q T}, B(t_q))$ are smaller than or equal to c_t and the slopes right to $(d_{t_q T}, B(t_q))$ are bigger than c_t.

STEP 2: Now g has to be convexified again, i.e., the slopes should be ordered. Since the breakpoints of g correspond to points $(d_{t_pT}, B(t_p))$ for $p=0,\ldots,P$, we just search for the maximal index r such that:

$$\frac{B(t_r)-B(t_{r-1})}{d_{t_r,t_{r-1}-1}} \leq \frac{B(t)-B(t_r)}{d_{t,t_r-1}} \tag{3.3.3}$$

Note that the minimality of t_r guarantees that g remains the lower envelope of the points $(d_{\tau T}, B(\tau))$, $\tau > t$. Since the left-hand side of (3.3.3) is equal to v_{t_r} we can rewrite this formula as $B(t_r)+v_{t_r}d_{t,t_r-1} \leq B(t)$. In other words t_r is chosen minimal such that $B(t_r)+v_{t_r}d_{t,t_r-1} \leq B(t)$.

Both t_q and t_r are found with exactly the same formulae as in the backward linear programming algorithm of section 2.3. Moreover, $B(t)$ is also calculated in the same way. It is therefore an easy exercise to prove that the breakpoints of g correspond to the potential periods, and that the slopes v_{t_p} $(p=0,\ldots,P)$ are exactly the dual variables of the backward linear programming algorithm. Hence, both algorithms are identical.

The complexity of the backward algorithm is similar to the complexity of the forward algorithm. However, there is a slight difference in the implementation of both. Due to the fact that the d_{1t} $(t=1,\ldots,T)$ are monotone, step 2 of the backward algorithm can be performed in $O(T)$ time, whereas the monotonicity of d_{1t} causes step 1 to take linear time in the forward algorithm. This implies that there is no need to implement the lower envelope with a 2-3 tree, in the backward algorithm. A simple stack suffices. Finally, if the production costs c_t are monotone, then step 1 can also be implemented in linear time. This implies the following theorem.

3.3.2. Theorem.

The backward algorithm can be implemented with a running time of $O(TlogT)$ using a stack to maintain the lower envelope. If the c_t $(t=1,\ldots,T)$ are monotone, then the backward algorithm runs in $O(T)$ time.

The geometric nature of the backward and forward algorithms is not only useful to get faster algorithms. The techniques enable us to prove some structural properties of (ELS) in a very elegant way. By way of illustration two examples of such properties are given. The first example is the planning horizon theorem of Wagner and Whitin, which is proved using the technique for the forward algorithm. The second example is a theorem on the structure of multiple optimal solutions, which is proved using the technique for the backward algorithm.

3.3.3. Theorem (*Planning horizon theorem of Wagner and Whitin*).

If, in an instance of (ELS), the production costs are monotone non−increasing, and if s is the last production period in an optimal schedule for the planning horizon $1,\ldots,t$, then the last production period for planning horizon $1,\ldots,t+1$ does not occur before s, i.e., the set $\{s,\ldots,t+1\}$ contains a production period.

Proof. Consider the lower concave envelope, after t iterations of the forward algorithm. The slopes of the line segments of g_t are in decreasing order because of the concavity of g_t. Since these slopes correspond to production costs of periods, the corresponding periods are in increasing order.

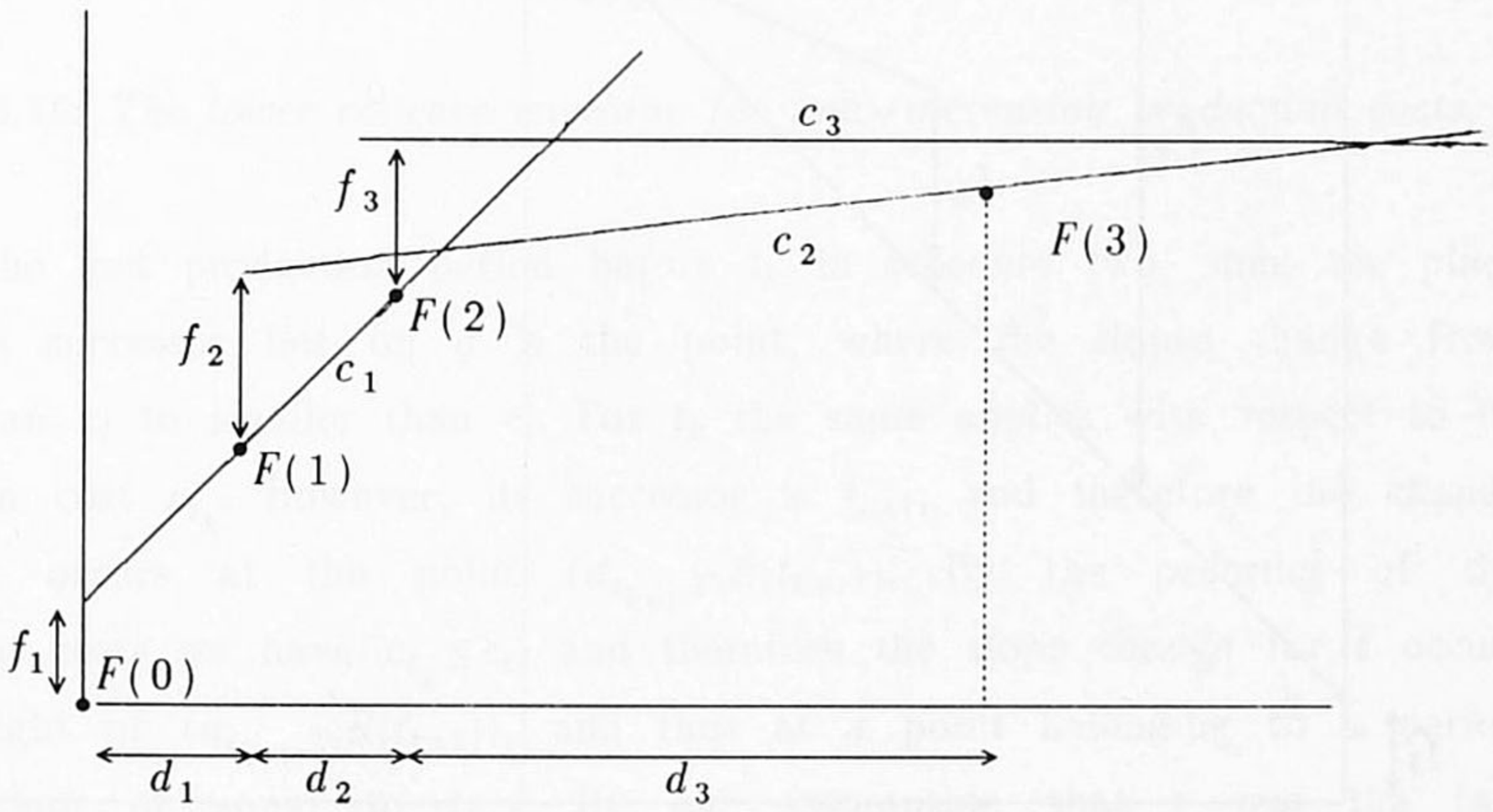


Figure 3.8. *The lower envelope for non−increasing production costs.*

Now suppose we have the lower envelope of the first t lines m_τ $(\tau = 0,\ldots,t-1)$. Then $F(t)$ is determined as the value of g_t at $x = d_{1t}$. Since s is the last production period in an optimal solution for the planning horizon $1,\ldots,t$ the line segment of g_t at $x = d_{1t}$ has slope c_s. Now the line m_t is added which has slope $c_t \leq c_s$. To determine $F(t+1)$ the value of g_{t+1} at $x = d_{1,t+1}$ is determined which occurs at right from d_{1t}. Due to the concavity of g_{t+1}, the slope of the line segment corresponding to $d_{1,t+1}$ is at most c_s, and thus by the monotonicity of the production costs the production period with production costs equal to the slope of that line segment cannot precede s.

□

In general theorem 3.3.3 is not true. A counterexample is given in the following table.

EXAMPLE

t	1	2	3	4	5
d_t	1	1	1	1	2
f_t	1	6	2	10	10
c_t	2	0	1	10	10

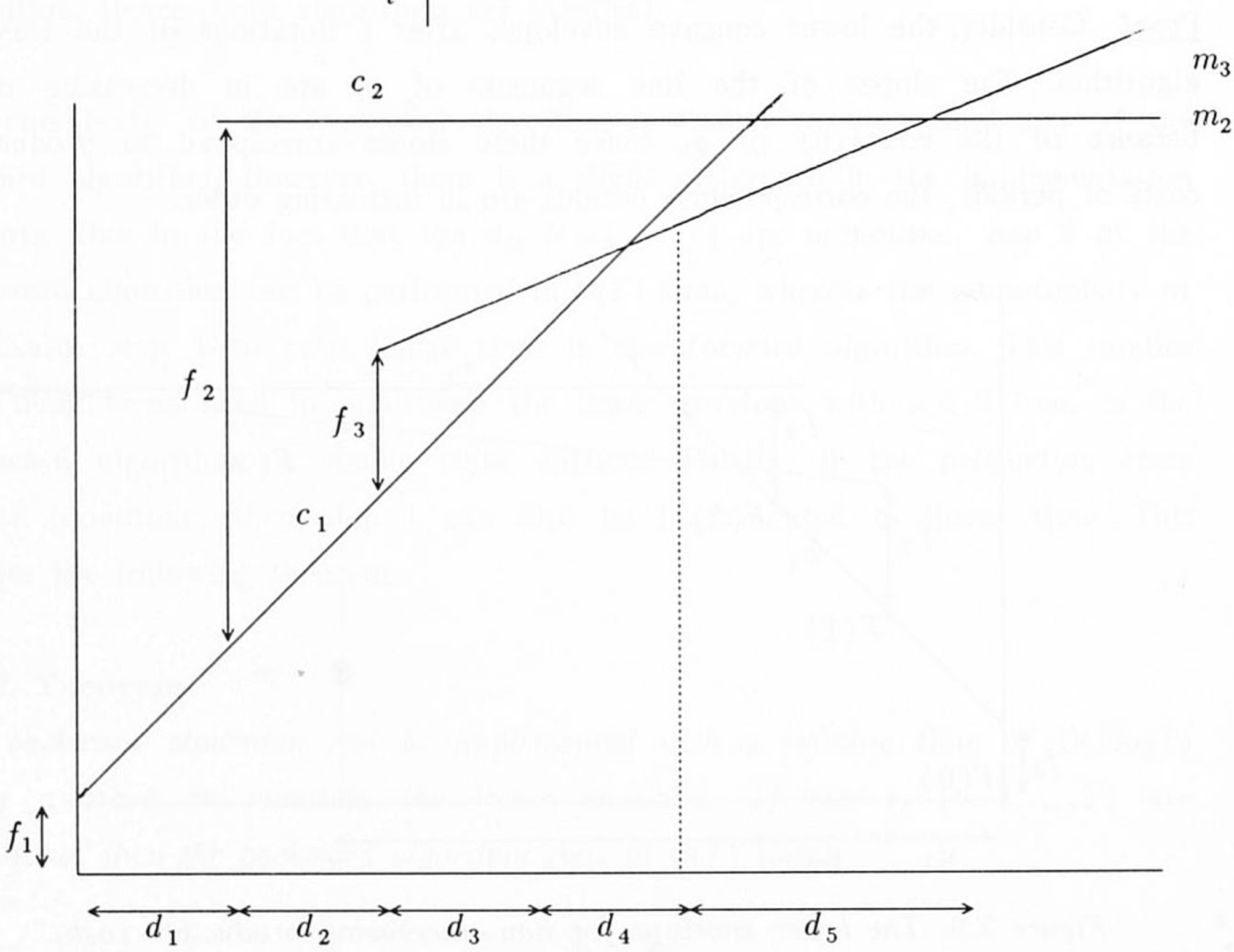


Figure 3.9. The lower concave envelope of the example.

A similar structural property concerns the relation between different optimal solutions.

3.3.4. Theorem.

Let an instance of (ELS) be given in which the production costs of the periods are non–increasing. Suppose that there are at least two optimal production schedules. Let the production periods of one optimal schedule be $1=t_1,\ldots,t_K$. *Then the second optimal schedule has a production period in each set* $\{t_k,\ldots,t_{k+1}\}$ *for* $k=1,\ldots,K-1$.

Proof Consider the final convex lower envelope of the backward algorithm and take the piece between $d_{t_k,T}$ and $d_{t_{k+1},T}$

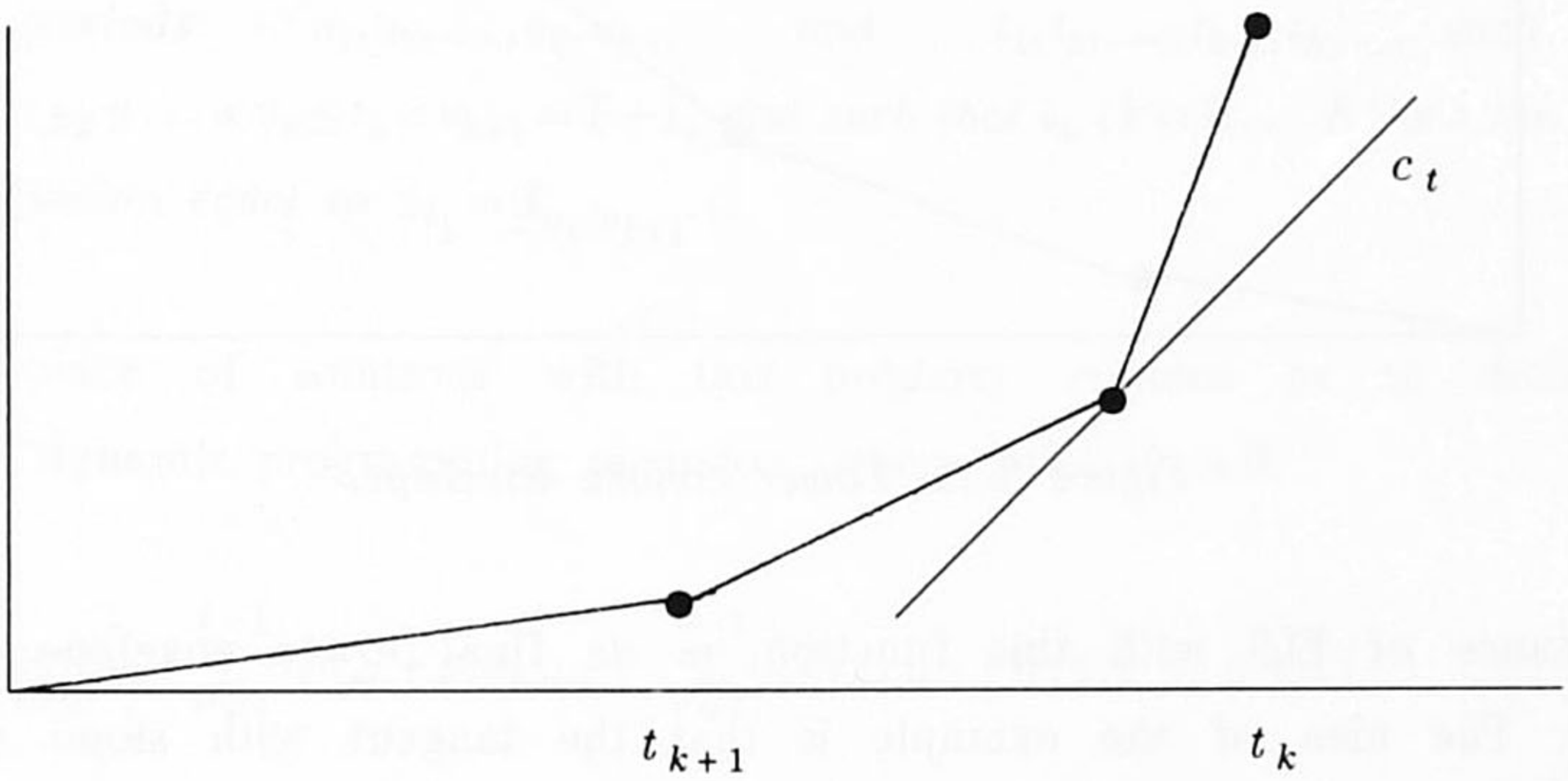


Figure 3.10. *The lower concave envelope for non–increasing production costs.*

If t is the last production period before t_k in schedule two, then the place where its successor lies on g is the point, where the slopes change from bigger than c_t to smaller than c_t. For t_k the same applies with respect to its production cost c_{t_k}. However, its successor is t_{k+1}, and therefore the change of slope occurs at the point $(d_{t_{k+1},T},B(t_{k+1}))$. By the ordering of the production costs we have $c_{t_k}\leq c_t$, and therefore the slope change for t occurs at the right of $(d_{t_{k+1},T},B(t_{k+1}))$, and thus at a point belonging to a period smaller than or equal to t_{k+1}. By our assumption that t was the last production period before t_k, this proves the theorem.

□

A counterexample for the general case is given by the following piecewise linear convex function.

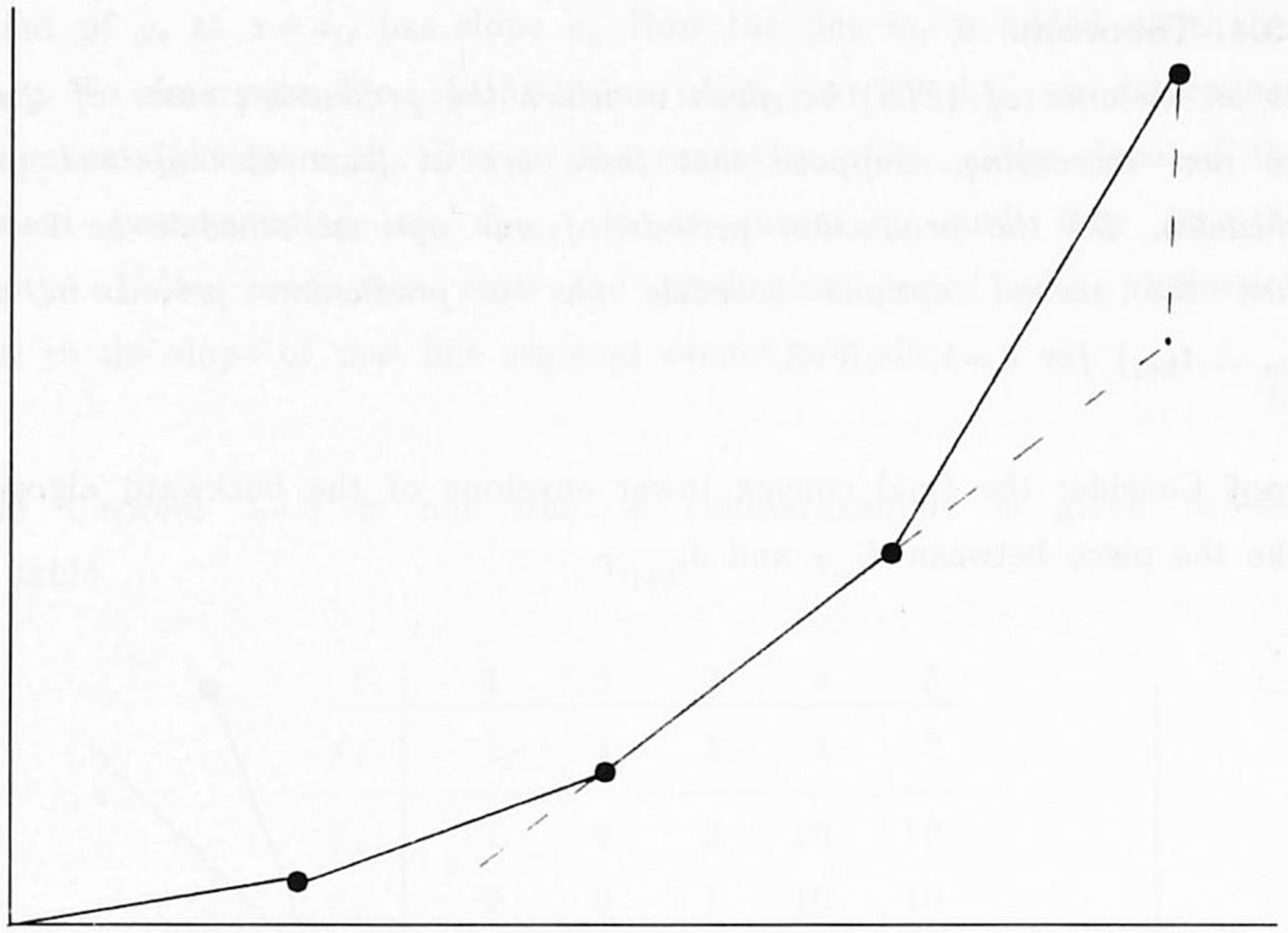


Figure 3.11. *Lower convex envelope.*

An instance of ELS with this function as its final lower envelope is easily derived. The idea of the example is that the tangent with slope c_1 passes through two consecutive breakpoints on the lower envelope. For the earliest period we generate very high set-up costs and zero production costs, so that there will be no later production periods. Finally, we make sure that there are more production periods after the second period. This is accomplished by the following example:

EXAMPLE

t	1	2	3	4
d_t	1	1	1	1
f_t	2	12	1	1
c_t	6	0	3	1

The optimal schedules have the following production periods: schedule 1: 1, 3 and 4; schedule 2: 1 and 2; both with value 20.

3.4. Applications to extensions of ELS.

The methods developed in section 3.2 can be applied to the extensions of ELS with backlogging and with start-up costs. The combinations of both extensions will also be shown to be solvable by the geometric techniques.

3.4.1. ELS with Backlogging

In chapter 1, the following version of the zero-inventory property for ELS with backlogging has been proved.

3.4.1. Theorem (Zero-Inventory property for ELSB).
There is an optimal solution to ELSB with the following characteristic. There exist periods $u_1, u_2, \ldots, u_k, u_{k+1}$ *and* $t_1, t_2, \ldots, t_{k-1}, t_k$ *such that* $1 \le u_1 \le t_1 < u_2 \le \ldots < u_k \le t_k < u_{k+1} = T+1$, *and such that* t_k $(k = 1, \ldots, K)$ *are the periods with production equal to* $x_{t_k} = d_{u_k, u_{k+1}-1}$.

The existence of solutions with this property enables us to define the following dynamic programming recursion, where $H(T+1) := 0$.

$$H(s) = \min_{s \le t < u} \{f_t + \sum_{\tau=s}^{t-1} h_\tau^- d_{s\tau} + p_t d_{s,u-1} + \sum_{\tau=t}^{u-1} h_\tau^+ d_{\tau+1,u-1} + H(u)\} \quad (1 \le s \le T) \qquad (3.4.1)$$

The variables $H(t)$ consist of the optimal production costs for the problem restricted to the planning horizon $t, \ldots, T$. This recursion was already proposed in a first paper on the problem by Zangwill [60]. He also proposed a straightforward procedure to solve this dynamic program in time proportional to T^3. In a later paper (Zangwill [61]), he improves upon this time bound to $O(T^2)$ by introducing new variables $H'(t)$, where $H'(t)$ is the cost of an optimal solution of ELSB restricted to the planning horizon $t, \ldots, T$, with the additional condition that t is a production period. Substituting $H'(t)$ in 3.4.1 gives the following double recursion, where $H'(T+1)$ and $H(T+1)$ are defined zero.

$$H(s)=\min_{t\geq s}\{p_t d_{s,t-1}+\sum_{\tau=s}^{t-1} h_\tau^- d_{s\tau}+H'(t)\} \qquad (1\leq s\leq T) \qquad (3.4.2)$$

$$H'(t)=\min_{u>t}\{f_t+p_t d_{t,u-1}+\sum_{\tau=t}^{u-1} h_\tau^+ d_{\tau+1,u-1}+H(u)\} \qquad (1\leq t\leq T) \qquad (3.4.3)$$

This double recursion is readily seen to be solvable in $O(T^2)$ time. The order in which the variables are determined is the following. If the variables $H'(t)$ and $H(t)$ are calculated for $t=s+1,\ldots,T$, then first $H'(s)$ is calculated, and then $H(s)$.

In order to show that both recursions are of the same type as the recursions 3.2.1 and 3.2.2, the following quantities are defined.

$$S(s,t) = \sum_{\tau=s}^{t-1} h_\tau^- d_{s\tau} \;; \qquad S'(t,u) = \sum_{\tau=t}^{u-1} h_\tau^+ d_{\tau+1,u-1}$$

Now the following formulas $S(s,t)=S(1,t)-S(1,s)-h_{s,t-1}^- d_{1,s-1}$, and $S'(t,u)=S'(1,t)-S'(1,u)-h_{1,t-1}^+ d_{t,u-1}$ hold. Note that these values can be calculated in constant time, if a linear time preprocessing is performed in which the quantities $h_{1,t}^-$, $h_{1,t}^+$, $d_{1,t}$, $S(1,t)$ and $S'(1,t)$ are calculated. The dynamic programming recursions 3.4.2 and 3.4.3 can be reformulated as follows.

$$H'(t)=f_t-S'(1,t)+\min_{u>t}\{H(u)+S'(1,u)+(p_t-h_{1,t-1}^+)d_{t,u-1}\};$$

$$H(s)=-S(1,s)+h_{1,s-1}^- d_{1,s-1}+\min_{t\geq s}\{H'(t)+p_t d_{1,t-1}+S(1,t)+(-h_{1,t-1}^- - p_t)d_{1,s-1}\}$$

These reformulations can be solved using the geometric techniques of section 3.2. The first recursion is of the type 3.2.2. It calculates $H'(s)$ from the values $H(t)$ $(t=s+1,\ldots,T)$ by use of the lower envelope of the points $(d_{tT},H(t)+S'(1,t))$. The second recursion calculates $H(s)$ from the values $H'(t)$ by use of the lower envelope of the lines through $(0,H'(t)+p_t d_{1,t-1}+S(1,t))$ with slope $-h_{1,t-1}^- - p_t$.

The following theorem is consequence of the results in section 3.2 and the preceding exposition.

3.4.2. Theorem. *ELSB is solvable in $O(TlogT)$ time.*

As argued in section 3.2, monotonicity conditions on the costs may speed up the algorithm to time proportional with T. Since d_{1t} is already monotone in t, the conditions in the lemmae 3.2.4 and 3.2.5 are translated into the conditions given in the following theorem.

3.4.3. Theorem.

ELSB is solvable in $O(T)$ time, if $p_t - h_t^- \leq p_{t+1} \leq p_t + h_t^+$, for $t \in \{1,\ldots,T-1\}$.

Proof. Searching on and updating the lower envelopes can be done in linear time if $p_t - h_{1,t-1}^+$ and $-h_{1,t-1}^- - p_t$ are monotone. Taking $p_t - h_{1,t-1}^+$ monotone non-increasing and $h_{1,t-1}^- + p_t$ monotone non-decreasing gives the condition mentioned in this theorem.

□

We have given a backward algorithm to solve ELSB. However, an algorithm based on a forward recursion acts essentially the same, since the forward double recursion is similar to the backward recursion.

3.4.2. ELS with start-up costs

The well-known Zero-Inventory property as it is known for the ordinary economic lot sizing problem is also valid for ELS with start-up costs, i.e., we may restrict ourselves to schedules, where production in any period equals the cumulative demand of the periods $t,\ldots,\tau$ for some $\tau \geq t$, and no production period occurs in the periods $t+1,\ldots,\tau$, To include the start-up costs in a dynamic program, two types of dynamic programming variables are defined.

$G(t)$: The minimum cost of a production plan that satisfies the demands of the periods $t,\ldots T$ in these periods. Note that the possible costs of a start-up and set-ups in periods before t are included.

The following variables do not take the start-up and set-up costs into account that are incurred in the periods $1,\ldots,t$. Note that the production costs in t are included.

$G'(t)$: The minimum cost of a production plan that satisfies the demands of the periods $t,\ldots T$ in these periods, where the start-up and set-up costs of the periods $1,\ldots,t$ are ignored.

The following recursions are easily seen to be a correct formulation of the problem.

$$G'(T+1):=0; G'(t)=\min_{\tau>t}\{c_t d_{t,\tau-1}+G(\tau), c_t d_{t,\tau-1}+f_{t+1,\tau}+G'(\tau)\}\} \quad (1\le t\le T)\ (3.4.4)$$

$$G(T+1):=0; \qquad G(t)=\min_{\tau\le t}\{g_\tau+f_{\tau t}\}+G'(t) \qquad (1\le t\le T)\ (3.4.5)$$

The recursion for $G'(t)$ may need a little more explanation. Here t and τ are considered to be consecutive production periods. The choice to be made is, whether it is cheaper to start-up production after t, in which case $\min_{\tau>t}\{c_t d_{t,\tau-1}+G(\tau)\}$ should be determined, or to perform set-ups in all periods between t and τ, in which case $-f_{1t}+\min_{\tau>t}\{c_t d_{t,\tau-1}+f_{1\tau}+G'(\tau)\}$ should be determined. The recurrence for $G'(t)$ is rewritten as the minimum of these two terms.

Since $G'(t)$ is the minimum over two terms, it is determined by using two lower envelopes; one of the points $(d_{\tau T}, G(\tau))$ and one of the points $(d_{\tau T}, f_{1\tau}+G'(\tau))$. It follows that $G'(t)$ can be determined in $O(T\log T)$ time for all $t\in\{1,\ldots,T\}$ simultaneously, and in $O(T)$ time if the production costs are monotone.

It is left to show how $G(t)$ is determined efficiently from $G'(t)$. In order to accomplish this, it suffices to show that the best start-up period for each period can be calculated in linear time. The crucial observation is that the best start-up periods $s(t)$ for all periods t form a non-decreasing sequence. For, suppose that s is the best start-up period for period $t-1$. Thus, $g_s+f_{s,t-1}=\min_{\tau\le t-1}\{g_\tau+f_{\tau,t-1}\}$. Adding f_t to both sides of the equation gives $g_s+f_{s,t}=\min_{\tau\le t-1}\{g_\tau+f_{\tau,t}\}$. Therefore, we only have to compare g_t+f_t with g_s+f_{st}. If $g_t+f_t\le g_s+f_{st}$, then t becomes the best start-up period, otherwise s will remain to be the best start-up period for t. It follows immediately that the best start-up period for each period can be determined in linear time, for all periods simultaneously. The following theorem is evident.

3.4.4. Theorem.
ELSS is solvable in $O(TlogT)$ time in general and in $O(T)$ time, if the production costs are monotone in t.

The techniques used by Klawe [26] and Aggarwal and Park [1] can also be used to generate the results of theorem 3.4.4, since the dynamic programs also fall in their class as discussed at the end of section 3.2.

3.4.3. ELS with Backlogging and Start-up costs.

The most natural generalization of the problems in subsections 3.4.1 and 3.4.2 is to combine them, i.e., backlogging is allowed and also start-up costs are included. The formulations of both separate models are easily combined. The solution techniques are a little more difficult to combine, although the techniques are used on separate parts of the model. Nevertheless, we will give the dynamic program for the combined problem. The variables are defined as follows.

$G(t)$: The cost of producing in periods $t,\ldots,T$ their demands with start-up costs and set-up cost of earlier periods included.

$G'(t)$: Like $G(t)$, but without start-up costs and set-up cost of earlier periods included.

$H(t)$: Like $G(t)$, but with t as the first production period.

$H'(t)$: Like $G'(t)$, but with t as the first production period.

Now $H'(s)$ can be calculated as the minimum of the following two terms:

$$\min_{t>s}\{f_s+p_s d_{s,t-1}+S'(s,t-1)+G(t)\} \quad \text{and} \quad \min_{t>s}\{f_{s,t-1}+p_s d_{s,t-1}+S'(s,t-1)+G\ (t)\};$$

$$H(s)=\min_{j\le s}\{g_j+f_{j,s-1}+H'(s)\};$$

$$G'(s)=\min_{t\ge s}\{f_{s,t-1}+p_t d_{s,t-1}+S(s,t-1)+H'(t)\};$$

$$G(s)=\min_{t\ge s}\{p_t d_{s,t-1}+S(s,t-1)+H(t)\}.$$

Here, $S(s,t)$ and $S'(s,t)$ are equal to the quantities defined in subsection 3.4.1. From the form of these recurrences it can be seen that the geometric techniques can be applied recursively, to determine all variables in time proportional to $T\log T$. The conditions that are sufficient to let the algorithm run in linear time, are equal to those mentioned in theorem 3.4.2.

3.5. Computational experiments.

Some limited testing has been performed on the algorithms for ELS, developed in this section. They are also compared to the algorithms that are known from the literature. From the complexity results in section 3.3, it is obvious that a distinction has to be made between instances for which the production costs are monotone and general instances. Therefore, this section is separated into two subsections, one subsection for each type of instances.

Algorithms for instances with monotone production costs

First, an overview will be given of the algorithms that have been tested.

B: The backward dynamic programming algorithm.
F: The forward dynamic programming algorithm.
M: The recursive matrix searching algorithm developed by Klawe.
W: The original dynamic programming algorithm of Wagner and Whitin, which uses the planning horizon theorem.

The instances on which the various programs were run, have been generated as follows. The set-up costs were fixed at 450. The holding costs were fixed at one, and the production costs were fixed at zero. Finally, the demands were taken 30 units in each period. These choices fix the fraction of production periods at one out of six.

The algorithms have been implemented in Turbo Pascal, and were run on an Olivetti M280 PC without co-processor. The results (running-time in seconds) are summarized in the table below.

T	B	F	W	M
500	0.2	0.3	0.2	0.5
1000	0.4	0.6	0.4	1.0
2000	0.8	1.2	0.9	2.0
4000	1.5	2.5	1.8	4.2
8000	3.0	5.0	3.6	8.5

Table 3.1. *Running-time (sec) of the $O(T)$ algorithms.*

The performance of all algorithms shows a linear relation with the length of the planning horizon. This is not surprising, for the algorithms B, F and K, since these are linear time algorithms. For algorithm W this is also not surprising, since it maintains the periods from the last production period on. This number was fixed at six in the instances that have been tested. However, this seems to imply that especially algorithm W is sensitive to variations in the fraction of production periods. In the following table the results are found in case the average number of periods for which a production period produces the demand is varied. This variation is created by varying the set-up costs. The instances on which the algorithms were tested had the same parameters as the instances of the first test. The planning horizon was fixed at 4000 periods.

f	number	B	F	W	M
50	2	1.5	2.5	0.9	2.6
200	4	1.5	2.5	1.3	3.8
450	6	1.5	2.5	1.8	4.3
800	8	1.5	2.5	2.2	4.4
1250	10	1.5	2.5	2.6	4.5
5000	20	1.5	2.5	4.8	4.8
20000	40	1.5	2.5	9.1	5.2

Table 3.2. *Influence of variations in the set-up costs.*

Although an influence on algorithm W is according to our expectations, it is rather strange that the matrix searching algorithm is also influenced by

variations in the set-up cost structure. However, this influence is much less than the influence on algorithm W.

Algorithms for instances with general production costs

An overview of the algorithms that have been tested, is given first.

B: The backward dynamic programming algorithm.
F: The forward dynamic programming algorithm.
M: The recursive matrix searching algorithm developed by Aggarwal and Park.
W: The original dynamic programming algorithm of Wagner and Whitin.

The instances used to test the performance of these general algorithms, were such that all parameters were generated completely random, but within prespecified non-negative bounds, and independent of each other.

T	B	F	M	W
500	0.2	0.8	1.8	9.1
1000	0.5	1.8	4.8	36.8
2000	1.0	3.8	14.1	141.0
4000	2.1	7.9	45.0	563.8
8000	4.4	16.9	151.6	

Table 3.3. Running-time (sec) of the general algorithms.

The results here are much more indicative on which choice is preferable. The bad behaviour of algorithm W is no surprise. However, the results of algorithm M are rather disappointing. Its behaviour is not really of the complexity as expected. This may be due to the complicated data manipulations that must be used to implement algorithm M. The same problem, but less dramatic, may be due to the fact that algorithm F is also outperformed by algorithm B. Note that algorithm F uses 2-3 trees to maintain the lower envelope of lines, whereas B is implemented with a stack. This points out another advantage of algorithm B, i.e., the simplicity with which it can be implemented. The only non-trivial part concerns a binary search.

Chapter 4

DYNAMIC PROGRAMMING ALGORITHMS FOR THE DISCRETE LOT SIZING AND SCHEDULING PROBLEM AND EXTENSIONS

4.1. Introduction

The dynamic programming algorithm for the discrete lot sizing and scheduling problem (DLS), defined in chapter 1, has a running time of $O(DT)$, where T is the length of the planning horizon, and D is the cumulative demand of the periods. In view of the techniques developed in the previous chapter, it is natural to ask whether such improvements can also be used te speed up the dynamic program for DLS. After all, DLS and ELS have some common characteristics: in a set of periods demand occurs which has to be fulfilled by production in earlier periods. Moreover, the costs concern a fixed component for starting up production, and a variable component for production of units of the item. However, there is also a rather unpleasant difference with ELS: production is limited to be binary in each period. The main problem with this type of constraints is that the zero-inventory property does not necessarily hold anymore. This can be seen from the following example, in which it is favourable to start production in a period with positive starting inventory.

EXAMPLE

t	1	2	3	4	5	6
d_t	0	1	0	0	1	1
g_t	0	1	1	0	1	1
c_t	0	0	1	0	1	1

Clearly, an optimal solution is such that production takes place in the periods 1, 2 and 4. However, at the end of period 3, the inventory is not zero.

The zero-inventory property for DLS is formulated as follows. A production batch is defined as a maximal set of consecutive production periods. A solution that satisfies the zero-inventory property is such that each production batch starts with a period with no starting inventory, or equivalently $z_t I_{t-1} = 0$ for all $t \in \{1, \ldots, T\}$. In section 4.3, conditions will be derived under which the zero-inventory property is valid.

If the zero-inventory property is valid, then the dynamic program for DLS can be reformulated. On this formulation one can apply geometric techniques to speed up the dynamic programming algorithms that calculate its solution. These techniques can be used to calculate the best solution that satisfies the zero-inventory property. Although these techniques are more involving than the techniques in chapter 3, it will turn out that there are some striking similarities with the results in that chapter. First, there are two slightly different techniques, one which solves the forward dynamic programming recursion, and one which solves the backward recursion. Second, these techniques are more powerfull, in case restrictions are imposed on the cost coefficients. Third, both techniques can be combined to solve the case with backlogging and set-up costs.

In section 4.2, the general form of the dynamic programming recursions that solve DLS, when the zero-inventory property holds, will be formulated. The two techniques to solve the recursions, a forward and a backward, are explained. Finally, some conditions under which these techniques run faster will be given. In section 4.3, these techniques will be used to solve DLS, DLSS and DLSB, i.e., the variants with set-up costs added and with a backlogging facility.

4.2. Geometric techniques

The dynamic programming recursions that are the subject of this section contain the following parameters. There are $2n$ numbers, n numbers denoted by A_i $(i=1,\ldots,n)$, and n numbers denoted by B_i $(i=1,\ldots,n)$. Finally, a binary n-vector d is given with components d_i $(i=1,\ldots,n)$.

The forward dynamic programming recursion is defined as follows.

$$G(0) := 0;$$
$$G(i) := \min_{j<i}\{G(j)+A_{j+1}+B_{j+1}+B_{j+2}+\ldots+B_{j+d_{j+1,i}}\} \qquad (i=1,\ldots,n) \quad (4.2.1)$$

This recursion is interpreted as follows. Between two indices $i, j+1$ $(i<j)$ the binary vector d contains $d_{j+1,i}$ ones. For all these ones simultaneously, a batch of indices is chosen starting with $j+1$. The cost of such a batch consists of the value B_k for each chosen period k, and a cost A_{j+1} for the first index in the batch. Therefore, the total cost of the batch is $A_{j+1}+B_{j+1}+B_{j+2}+\ldots+B_{j+d_{j+1,i}}$. The variables $G(i)$ can be defined as follows. The variables $G(i)$ are the minimum costs of partitioning the set of indices $\{1,\ldots,i\}$ into subsets $\{i_r+1,\ldots,i_{r+1}\}$ of consecutive periods, such that the first $d_{i_r+1,i_{r+1}}$ periods in this subset are chosen $(r=1,\ldots,R)$. The costs involved with these choices are $A_{i_r+1}+B_{i_r+1}+\ldots+B_{i_r+\alpha_r}$, where $\alpha_r = d_{i_r+1,i_{r+1}}$.

The algorithm to solve the recursion 4.2.1 relies on the following geometrical interpretation of the values $G(j)+A_{j+1}+B_{j+1}+B_{j+2}+\ldots+B_{j+d_{j+1,i}}$ $(j=0,\ldots,i-1)$. The points $(d_{1j}, G(j)+A_{j+1})$ are drawn in the plane. From each point $(d_{1j}, G(j)+A_{j+1})$, a piecewise linear function $f_j(x)$ is defined, with $d_{j+1,i}$ linear pieces, until the line with x-coordinate d_{1i}. The k-th linear piece lies in the interval $[d_{1,j}+k-1, d_{1,j}+k]$ and its slope is B_{j+k}.

EXAMPLE

i	1	2	3	4	5	6
d_i	0	1	1	0	1	1
A_i	2	1	3	1	7	7
B_i	1	0	4	0	0	1

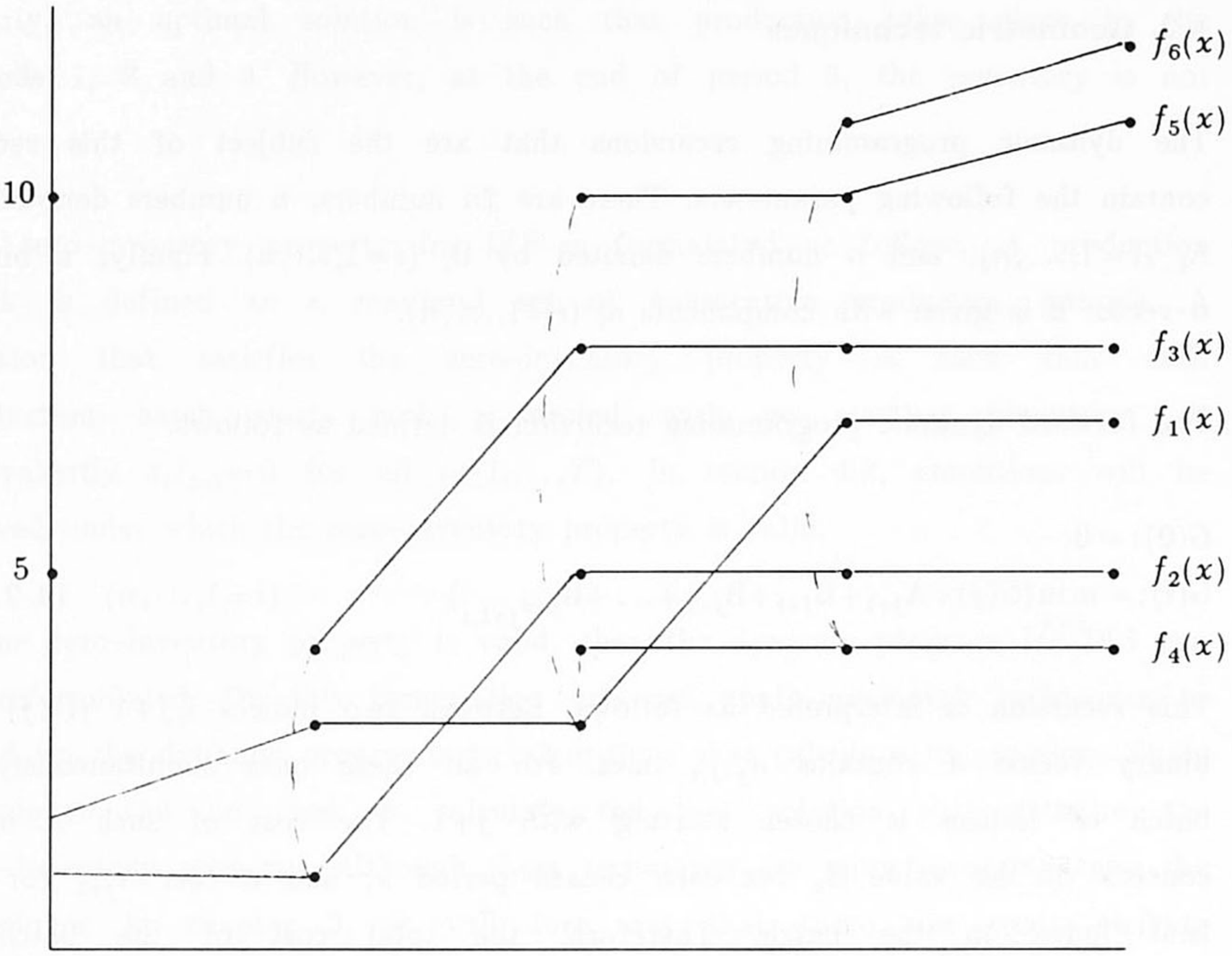


Figure 4.1. *Geometric interpretation of the forward dynamic program.*

In an iteration of the algorithm, it is easy to determine $G(i)$. It is the minimum value of the piecewise linear functions $f_j(x)$ $(j=0,\ldots,i-1)$ at coordinate $x=d_{1i}$. After this minimum is determined, the point $(d_{1i},G(i)+\mathrm{A}_{i+1})$ can be added. The remainder of the iteration consists of updating the piecewise linear functions. In case $d_{i+1}=0$, there is nothing left to do, since no extra index is chosen. We determine the minimum value of the piecewise linear functions at coordinate $d_{1,i+1}=d_{1i}$. In case $d_{i+1}=1$, to each function another line segment is added, i.e., to the function corresponding to index j the piece between d_{1i} and $d_{1,i+1}=d_{1,i}+1$ is added with slope $\mathrm{B}_{j+d_{j+1,i+1}}=\mathrm{B}_{j+d_{j+1,i}+1}$.

The complexity of this algorithm can be determined quite easily. Suppose that in the i-th iteration the minimum of all the piecewise linear functions is determined. Then, if $d_{i+1}=0$, a constant amount is needed for the addition of $(d_{1i},G(i)+\mathrm{A}_{i+1})$. If $d_{i+1}=1$, then i line pieces are added to the functions

$f_j(x)$ $(j=0,\ldots,i-1)$. At the same time the minimum of these functions at $x=d_{1,i+1}$ can be determined. Therefore, there are only $D:=d_{1,n}$, iterations in which $O(n)$ operations have to be performed. In the other iterations, a constant number of steps must be performed. Thus, the following theorem has been proved.

4.2.1. Theorem.

The dynamic programming recursion 4.2.1 *can be solved in* $O(Dn)$ *time.*

If restrictions are imposed on the cost coefficients, further improvements in this running time can be achieved. The major improvement can be gained by restricting the B_i to be non-increasing. Clearly, the piecewise linear functions $f_j(x)$ are concave under this restriction. Now compare the values of the functions corresponding to the indices j_1 and j_2, with respect to an arbitrary higher index i, i.e., the value $f_{j_1}(d_{1i})$, which equals $G(j_1)+A_{j_1+1}+B_{j_1+1}+B_{j_1+2}+\ldots+B_{j_1+d_{j_1+1,i}}$ and the value $f_{j_2}(d_{1i})$, which is $G(j_2)+A_{j_2+1}+B_{j_2+1}+B_{j_2+2}+\ldots+B_{j_2+d_{j_2+1,i}}$. If $f_{j_2}(d_{1i})$ is not higher than $f_{j_1}(d_{1i})$, then this is called "j_2 *dominates* j_1 *with respect to* i", or "j_2 *is* $i-$*dominant over* j_1". The geometric interpretation of i-dominance is that the function corresponding to j_2 lies below the function corresponding to j_1 at d_{1i}, if j_2 is i-dominant over j_1. The use of i-dominance follows from the following lemma and its corollary.

4.2.2. Lemma.

Suppose that the B_j $(j=1,\ldots,n)$ *are monotone non*$-$*increasing. Let* $j_1<j_2$. *If* j_2 *is* $i-$*dominant over* j_1, *then* j_2 *is also* $(i+1)-$*dominant over* j_1.

Proof. The following two inequalities are given. $f_{j_1}(d_{1i})\geq f_{j_2}(d_{1i})$, since j_2 is i-dominant over j_1, and $B_{j_1+d_{j_1+1,i+1}}\geq B_{j_2+d_{j_2+1,i+1}}$. The latter follows from the fact that the B_j are non-increasing, and $j_1+d_{j_1+1,i+1}\leq j_2+d_{j_2+1,i+1}$, due to the fact that the demand is is 0/1 function. If $d_{i+1}=0$, then trivially $f_j(d_{1,i+1})=f_j(d_{1,i})$ for each $j<i$. Otherwise, if $d_{i+1}=1$, then $f_j(d_{1,i+1})=f_j(d_{1,i})+B_{j+d_{j+1,i+1}}$ for each $j<i$. Therefore, in both cases it follows that $f_{j_1}(d_{1,i+1})\geq f_{j_2}(d_{1,i+1})$

□

Note that the functions $f_j(x)$ and $f_{j+1}(x)$ are parallel, if $d_{j+1}=1$. Therefore, it suffices to determine the minimum of both functions once, at $x=d_{1,j+1}$.

It is obvious that the minimum in 4.2.1 is attained by an index that i-dominates all other indices. From lemma 4.2.2, it follows that, if j_1 is i-dominated by j_2 $(j_1<j_2)$, then j_1 need not be considered for indices $i+1,\ldots,n$. Therefore, it suffices to keep an ordered list of indices $0\le j_0<j_1<\ldots<j_K\le j$ such that each index j not in the list is i-dominated by some index $j_k>j$. It follows that the list is built up such that j_{k+1} is i-dominated by j_k (otherwise j_k would have been removed from the list). The desired index in the list, concerning the minimum in 4.2.1, is therefore simply j_0.

These observations are not only useful for finding the optimal index in the dynamic programming recursion, but also to decrease the number of indices in the list. From lemma 4.2.2, it follows directly that for each pair of indices j_1,j_2, there is a so-called turning index, where dominance changes, as is stated in the following corollary.

4.2.3. Corollary.

Suppose that the B_j $(j=1,\ldots,n)$ *are monotone non-increasing. For each pair of indices* j_1 *and* j_2 $(j_1<j_2)$, *there is an index* u, *such that for* $i\in\{j_2,\ldots,u\}$ *we have* j_1 *is* i*-dominant over* j_2, *and for* $i\in\{u+1,\ldots,n\}$ *we have* j_2 *is* i*-dominant over* j_1.

Proof. Trivial from lemma 4.2.2.

□

Let the turning indices of the pairs of indices (j_{k-1},j_k) be u_k $(k=1,\ldots,K)$. If $u_k\ge u_{k+1}$ for some k, then j_k will NEVER become first in the list, because j_{k+1} dominates j_k with respect to the indices $\{u_{k+1}+1,\ldots\ldots,n\}$, and j_{k-1} dominates j_k with respect to the indices $\{j_k,\ldots,u_k\}$. Therefore, it can be removed from the list. The resulting list of indices consists of, say, $0\le j_0<j_1<\ldots<j_K\le i$, where the turning indices u_k for pairs (j_{k-1},j_k) are monotonically increasing. The indices in this list will be called potential indices. The iteration where $G(i)$ is determined, consists of the following two

steps. Let the following list of potential indices $j_0 < j_1 < \ldots < j_K$ with turning indices $u_1 < u_2 < \ldots < u_K$ be given.

STEP 1: The optimal preceding index for i is the first index in the list of potential indices, i.e., the index j_0. $G(i)$ can now be determined, and $(d_{1i}, G(i) + A_{i+1})$ can be added to the set of piecewise linear concave functions f_{j_k}.

STEP 2: First, if $u_1 = i$, then j_0 is removed from the set of potential indices. Second, the index i is added to the list of potential indices, and the turning index for the periods (j_K, i) is determined, say u_{K+1}. If this destructs the monotonicity of the turning indices in the list, i.e., if $u_K \geq u_{K+1}$, then j_K is removed from the list. This process repeats itself, until the monotonicity of the turning periods is recovered.

The determination of a turning index u can be done by binary search with the test that compares $f_j(d_{1u})$ with $f_i(d_{1u})$ for two given indices j and i, until the desired index u is found. Note that a preprocessing step to calculate B_{1u} and d_{1u} for each u, must be performed to let this test run in constant time. Thus, the complete test takes $O(\log n)$ time, leading to the following theorem.

4.2.4. Theorem.

The dynamic programming recursion 4.2.1 *can be solved in* $O(n \log n)$ *time, if the costs* B_i *are non-increasing.*

If the costs A_i are also non-increasing, it follows easily that an optimal solution exists in which all batches start at indices i with $d_i = 1$. For suppose that a batch starts at j with $d_j = 0$. Then the batch can be shifted one position higher to start at index $j+1$, without a cost increase. Using this repeatedly until an index i with $d_i = 1$ is encountered proves this. It follows that we can restrict the previous analysis to the D indices for which the component of the d-vector is non-zero. Taking into account a linear time preprocessing step, this leads to the following theorem for the case, where all costs are non-increasing.

4.2.5. Theorem.

The dynamic programming recursion 4.2.1 *can be solved in* $O(n+DlogD)$ *time, if the costs* A_i *and* B_i $(i=1,\ldots,n)$ *are non-increasing.*

The remainder of this section is devoted to the analog of the backward algorithm of section 3.2. The backward dynamic programming recursion is defined as follows.

$$G(n+1) := 0;$$

$$G(i) := \min_{j>i}\{A_i+B_i+B_{i+1}+\ldots+B_{i-1+d_{i,j-1}}+G(j)\} \qquad (i=1,\ldots,n) \quad (4.2.2)$$

The algorithm to solve the recursion 4.2.2 can also be interpreted geometrically. Piecewise linear functions $f_j^i(x)$ that represent the values $G(j)+B_i+B_{i+1}+\ldots+B_{i-1+d_{i,j-1}}$ $(j=i+1,\ldots,n+1)$ are defined as follows. For a given j $(j=i+1,\ldots,n+1)$, the points $(d_{jn},G(j))$ are drawn in the plane. The $d_{i,j-1}$ linear pieces on the intervals $[d_{in}-k, d_{in}-k+1]$ for $k=1,\ldots,d_{i,j-1}$, have a slope of B_{i+k}.

The slopes of the functions $f_j^i(x)$ are only dependent on i and the interval. Thus, they are NOT dependent on j. This follows from the fact that this slope is B_{i+k} on the interval $[d_{in}-k, d_{in}-k+1]$. These items depend on i and k only. Now, $G(i)$ is the minimum value of the piecewise linear functions $f_j^i(x)$ $(j=i+1,\ldots,n+1)$ at coordinate $x=d_{1i}$. After this minimum is determined the point $(d_{in},G(i))$ can be added. It is important to note that, in case $d_i=0$, then it suffices to compare $G(i)$ with $G(i+1)$. Only the function corresponding to the lowest value of the two needs to be maintained, in the algorithm. Therefore, the number of lines will never exceed $D=d_{1n}$.

EXAMPLE

i	1	2	3	4	5	6
d_i	0	1	1	0	1	1
A_i	2	1	2	4	4	1
B_i	3	0	4	0	1	1

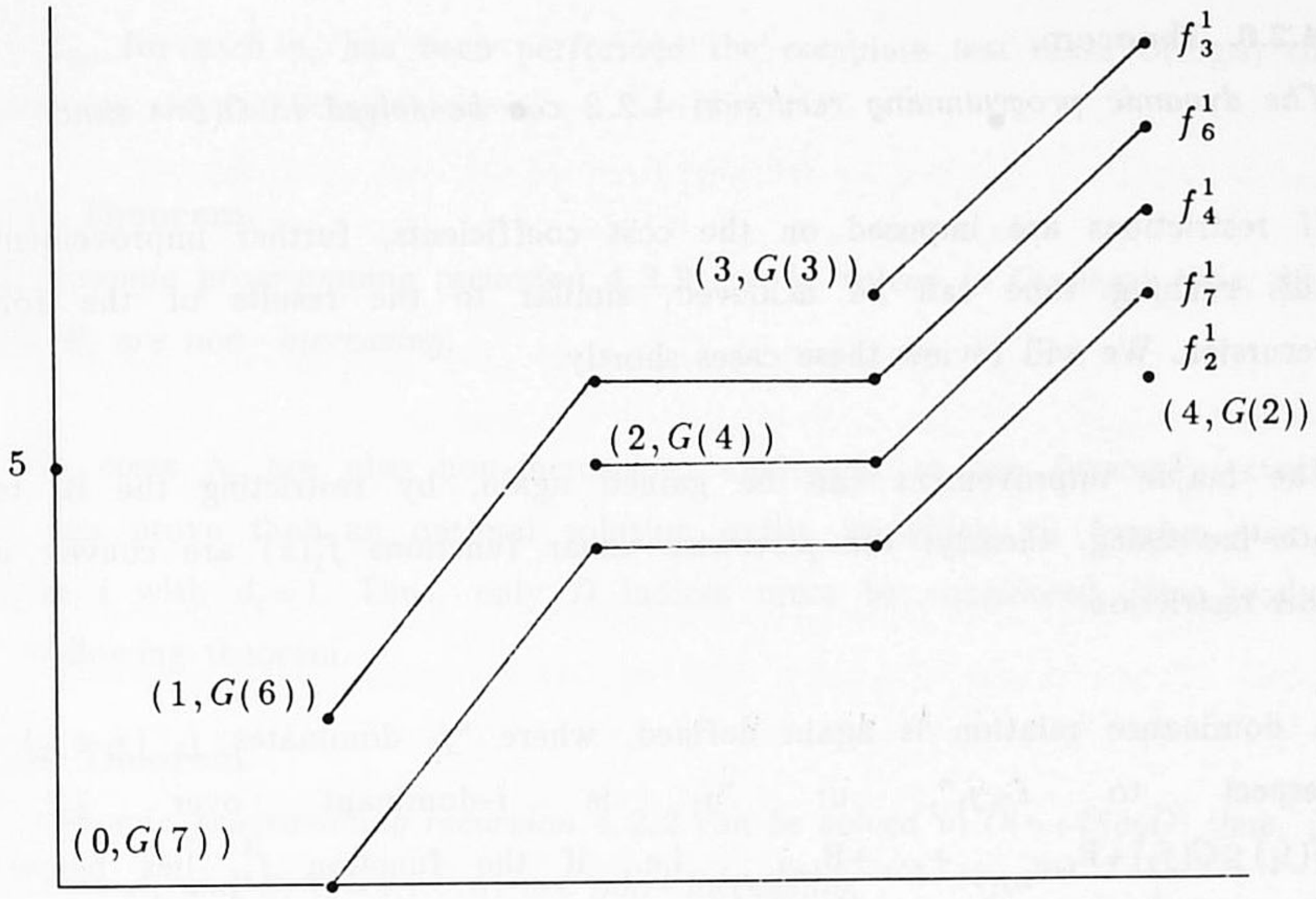


Figure 4.2. Geometric interpretation of the backward dynamic program.

The remainder of the iteration consists of updating the piecewise linear functions. In case $d_{i-1}=1$, then to each function f_j^i the piece on the interval $[d_{i,n}, d_{i-1,n}]$ with slope B_{i-1} is added. Therefore, the relation between each pair of functions is not changed. The case $d_{i-1}=0$ is the difficult case. Here, for each j the part of the recursion consisting of the sum $B_i+B_{i+1}+\ldots+B_{i-1+d_{i,j-1}}$ should be replaced by $B_{i-1}+B_i+\ldots+B_{i-2+d_{i-1,j-1}}$. This implies that B_{i-1} should be added, and $B_{i-1+d_{i,j-1}}$ should be subtracted. The addition of B_{i-1} can easily accomplished by adding a line piece with this slope to the right of the function. However, subtracting $B_{i-1+d_{i,j-1}}$ involves a shift of the function corresponding to j to the left of one unit. This may destroy the order of the functions, since that order depends on j.

Nevertheless, the complexity of the algorithm can be determined quite easily. Suppose that in the iteration, where $G(i)$ is determined, the minimum of all the piecewise linear functions must be calculated. The number of calculations is bounded by the number of functions, which does not exceed D, since there are at most D starting points for functions. Therefore, the following theorem is evident.

4.2.6. Theorem.

The dynamic programming recursion 4.2.2 *can be solved in* $O(Dn)$ *time.*

If restrictions are imposed on the cost coefficients, further improvements in this running time can be achieved, similar to the results of the forward recursion. We will review these cases shortly.

The major improvement can be gained again, by restricting the B_i to be non–increasing. Clearly, the piecewise linear functions $f_j^i(x)$ are convex under this restriction.

A dominance relation is again defined, where "j_1 dominates j_2 $(j_1 < j_2)$ with respect to $i < j_1$", or "j_1 is i–dominant over j_2", if $G(j_1) \leq G(j_2) + B_{i+d_{i,j_1+1}} + \ldots + B_{i+d_{i,j_2}}$, i.e., if the function $f_{j_1}^i$ lies below the function $f_{j_2}^i$. Otherwise, j_2 is i–dominant over j_1.

We can again construct a list of potential indices consisting of say $n+1 \geq j_0 > j_1 > \ldots > j_K > i$, with the turning indices u_k for pairs (j_{k-1}, j_k) monotonically decreasing. This results in the following steps for the iteration, where $G(i)$ is determined.

STEP 1: The optimal succeeding index for i is now the first index in the list of potential indices, the index j_0. $G(i)$ can now be determined, and $(d_{in}, G(i))$ is added to the set of piecewise linear convex functions.

STEP 2: First, if $u_1 = i$, then j_0 is removed from the set of potential indices. Second, the turning index for the periods (j_K, i) is determined, say u_{K+1}. If this destructs the monotonicity of the turning indices in the list, i.e., if $u_K \geq u_{K+1}$, then j_K is removed from the list. This process repeats itself, until the monotonicity of the turning periods is recovered.

The determination of a turning index u can be done by binary search with the test that compares $f_j^u(d_{un})$ with $f_i^u(d_{un})$ for two given indices j and i, until the desired index u is found. Given that a preprocessing step to calculate B_{un}

and d_{un} for each u, has been performed the complete test takes $O(\log n)$ time, leading to the following theorem.

4.2.7. Theorem.

The dynamic programming recursion 4.2.2 *can be solved in* $O(n\log n)$ *time, if the costs* B_i *are non-increasing.*

If the costs A_i are also non-increasing, analogous to the forward recursion, one can prove that an optimal solution exists in which all batches start at indices i with $d_i = 1$. Thus, only D indices must be considered. This leads to the following theorem.

4.2.8. Theorem.

The dynamic programming recursion 4.2.2 *can be solved in* $O(n+D\log D)$ *time, if the costs* A_i *and* B_i $(i=1,\ldots,n)$ *are non-increasing.*

It has been shown how a geometrical interpretation of the dynamic programs 4.2.1 and 4.2.2 leads to a speed up of the solution algorithm. This speed up is even better, if a dominance relation can be derived, based on a monotonicity property of the cost coefficients. Although the results in this section may not look too impressive, it is shown in the following section that they can be used to improve the dynamic programming algorithms for several versions of DLS.

4.3. Discrete Lot Sizing and Scheduling Problems

Clearly, the dynamic programming recursion for DLS as defined in section 1.3, can be fit into the types of dynamic programs of section 4.2. This is done by identifying the start-up costs g_t with the A_i, and by identifying the production costs c_t with the B_i. Applying the results of section 4.2 to these dynamic programs, we get the following theorems immediately.

4.3.1. Theorem.

The best schedule that satisfies the zero-inventory property, for a general instance of DLS can be found in $O(DT)$ *time, where* D *is the cumulative demand.*

A sufficient condition, for which the zero-inventory property holds, is that the production costs are non-increasing in t. This is exactly the condition which is necessary to speed up the algorithm, as the following theorem claims.

4.3.2. Theorem.

DLS can be solved in $O(TlogT)$ time, if the production costs c_t are non-increasing.

4.3.3. Theorem.

DLS can be solved in $O(DlogD)$ time, if the set-up and production costs g_t and c_t are non-increasing, provided that an $O(T)$ preprocessing to calculate the c_{1t} has been performed.

Finally, if the set-up, production and holding costs do not depend on the periods, i.e., $g_t = g$, $p_t = p$ and $h_t = h$, two implications follow which make the implementation even simpler:

1) Set-ups only take place at demand periods, where the previous period is a non-demand period, i.e. at the beginning of a so-called demand batch. This is shown by an easy exchange argument.

If the number of demand batches is denoted by D_0, then the number of periods to be considered in the analysis reduces to D_0.

2) The turning period for two indices i_1 and i_2 $(i_1 < i_2)$, can directly be calculated as the largest period τ smaller than the current period, for which

$$p(i_2 - i_1) + h \sum_{i=0}^{i_2 - i_1 - 1} (T - \tau - 1) \geq G(i_2) - G(i_1)$$

Since τ can be calculated in constant time from this formula we get the following theorem.

4.3.4. Theorem.

If g_t, p_t, h_t are constant over all periods, then DLS can be solved in $O(D_0)$ time, where D_0 is the number of demand batches.

Similar results are derived for the extension of DLS, where set-up costs are involved, denoted by DLSS. In order to derive a Zero-Inventory property for DLSS the notion of set-up batches is introduced. A set-up batch is a maximal set of consecutive periods, in which set-ups are performed. The first period in the batch is therefore a start-up period.

In an optimal solution the first production period in a set-up batch should be a period with zero starting inventory. A sufficient condition to achieve this is that f_t+c_t are monotone non-increasing as a function of t. Indeed if we take two consecutive set-up batches B_1 and B_2 such that a feasible schedule has positive starting inventory at the first production period t_1 in B_2 then we can use the first non-production period t_2 after t_1 to take over the production of the last production period in B_1. By the monotonicity of f_t+c_t this cannot increase the cost of the schedule. Note that if the monotonicity of f_t+c_t does not hold, then it might be profitable to have non-zero starting inventory in a production period.

EXAMPLE

t	1	2	3	4	5
d_t	0	0	0	1	1
g_t	0	1	0	1	1
f_t	0	1	0	1	1
c_t	0	1	0	1	1

The optimal schedule produces in the periods 1 and 3 to satisfy the demand of the periods 4 and 5. Thus the starting inventory of the set-up batch {3} equals 1.

Unfortunately, monotonicity of f_t+c_t is not enough to ensure that any production batch (there may be more than one production batches in a set-up batch) starts with no inventory.

EXAMPLE

t	1	2	3	4	5
d_t	0	0	0	1	1
g_t	0	10	10	10	10
f_t	5	0	3	2	1
c_t	0	4	0	0	0

In this example the optimal schedule consists of one set-up batch {1,2,3} and two production batches {1} and {3}. Again the starting inventory of period 3 is not zero. A sufficient condition to overcome this is that the production costs c_t are non-increasing too. If there are two consecutive production batches P_1 and P_2 in one set-up batch in a feasible solution under this condition then we can shift the last production period of P_1 to just before P_2 at no extra cost. This leads to the following version of the zero-inventory property:

4.3.5. Theorem (Zero-Inventory property for DLSS).
If the c_t and f_t+c_t are monotone non-increasing with respect to t, then there is an optimal solution to DLSS, where all production batches have zero starting inventory.

Now let $G(t)$ be the value of the optimal solution for the planning horizon $(t,\ldots,T)$, where the set-up and start-up costs, possibly of periods preceding t, are included. $G'(t)$ is like $G(t)$, but only with set-up costs from t on, included. Then, the following dynamic programming recursion can be defined.

$$G'(t) = \min\{\min_{\tau > t}\{f_{t,t+d_{t,\tau-1}} + c_{t,t+d_{t,\tau-1}} + G(\tau)\},\ \min_{\tau > t}\{f_{t,\tau-1} + c_{t,t+d_{t,\tau-1}} + G'(\tau)\}\}$$

$$G(t) = \min_{s \le t}\{g_s + f_{s,t-1}\} + G'(t).$$

To derive $G'(t)$ with the techniques of section 4.2, we split the calculations for $G'(t)$ into two parts in the obvious way. For the first part, it suffices to use the technique for the backward algorithm of section 4.2, where B_i is replaced by f_t+c_t, and for the second part, the same technique is used but now with B_i replaced by c_t. This gives the following theorem.

4.3.6. Theorem.

DLSS can be solved in $O(T\log T)$ time provided that c_t and f_t+c_t are monotonically non-increasing in t. If the g_t are also non-increasing, then an algorithm with complexity $O(T+D\log D)$ can be derived.

For the backlogging case, the following double recursion can be derived. Here, the variables $G(t)$ $(t=1,\ldots,T)$ denote the optimal solution to the problem, restricted to the planning horizon $t,\ldots,T$. The variables $G'(t)$ $(t=1,\ldots,T)$ denote the optimal solution to the problem, where t is a start-up period.

$$G(t)=\min_{\tau\geq t}\{g_{\tau-d_{t,\tau-1}}-g_\tau+p_{\tau-d_{t,\tau-1}}+\ldots+p_{\tau-1}+G'(t)+BC(\tau,t)\}$$

$$G'(t)=\min_{\tau>t}\{g_t+p_t+\ldots+p_{t+d_{t,\tau-1}-1}+G(t)+HC(\tau,t)\}$$

Here, $BC(\tau,t)$ denote the backlogging costs, and $HC(\tau,t)$ denote the holding costs. This double recursion fits in the dynamic programs of section 4.2, analogous to the way that the recursions of ELSB are handled, with respect to the dynamic programs applicable to ELS. Therefore, an $O(DT)$ algorithm is evident. However, to speed up the algorithm to an $O(T\log T)$ algorithm very severe restrictions on the cost coefficients are necessary. A carefull analysis leads to the following restrictions.

i) $g_t-g_{t+1}+p_t+h^+_{tT}$ non-increasing in t;

ii) $g_t-g_{t+1}+p_t+h^+_{tT}+h^-_{1t}$ non-decreasing in t.

Chapter 5

SENSITIVITY ANALYSIS OF THE ECONOMIC LOT SIZING PROBLEM

5.1. Introduction

In this chapter we study sensitivity analysis problems of ELS. In particular we are concerned with the computation of the maximal ranges in which the numerical problem parameters may vary individually, such that a solution already obtained remains optimal. These computations highly rely on the concepts developed in chapter 3, for solving ELS quickly.

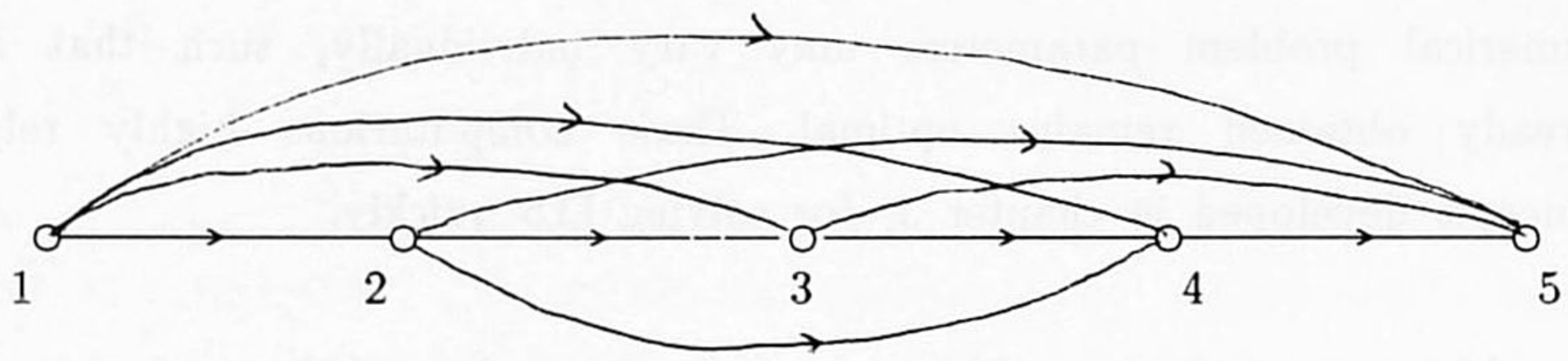
The history on these problems is very short. Lee [29] presents a theoretical framework to perform a similar analysis on general dynamic programming problems. He shows how his general dynamic programming methods are applicable to the economic lot sizing problem, but he does not consider the computational aspects of his approach. The basic concept of his work concerns the construction of a so-called penalty network. For the economic lot sizing problem, this construction requires already $\Omega(T^2)$ time, while most of our algorithms have a lower running time. Parametric analyses for various special cases of ELS have been considered by Richter [42], Richter and Vörös [43] and [44], and van Hoesel and Wagelmans [22]. For a detailed survey on parametric analysis in general, see Wagelmans [51].

This chapter is organized as follows. In Section 5.2, we prove some preliminary results. The actual sensitivity analysis is performed in section 5.3. Section 5.3.1 treats the analysis of the set-up costs and the production costs. Section 5.3.2 considers the holding costs, and section 5.3.3 focuses on the demands. Finally, section 5.4 contains concluding remarks.

5.2. Preliminary results for the sensitivity analysis of ELS

This section contains some lemmas, which are useful in the following section, where the actual sensitivity analysis is performed. The concepts make extensive use of the dynamic programming variables $F(t)$ and $B(t)$ $(t=1,\ldots,T)$. Therefore, the dynamic programming network for ELS is useful in the exposition, although it is never actually constructed in the algorithms to be presented. It is given below, together with the dynamic programming recursions.

The vertex set is $\{1,2,\ldots,T+1\}$; the set of arcs is $\{(s,t)|1\leq s<t\leq T+1\}$ and the length of arc (s,t) is equal to $l_{st}\equiv f_s+c_s d_{s,t-1}$.



$$l_{24}\equiv f_2+c_2 d_{23}$$

Figure 5.1. *Dynamic programming network for* $T=4$.

The recursions are

Forward: $F(0):=0;$ $F(t):=\min_{\tau\leq t}\{F(\tau-1)+f_\tau+c_\tau d_{\tau t}\}$ $(t=1,\ldots,T)$

Backward: $B(T+1):=0;$ $B(t):=\min_{\tau>t}\{f_t+c_t d_{t,\tau-1}+B(\tau)\}$ $(t=T,\ldots,1)$

The length of a shortest path between 1 and t $(t=1,\ldots,T)$ is denoted by $F(t-1)$, and the length of a shortest path from t to $T+1$ in this network is equal to $B(t)$. In the sequel the notions of "path" and "production plan", as well as "length" and "cost" will be used as synonyms. It should be noted that the dynamic program does not allow so-called dummy set-ups, which might be favourable, if the set-up costs are negative and the unit production are high. Therefore, we exclude this by disallowing negative set-up costs.

The following three lemmas concern basic facts with respect to the costs of partial production plans. Lemma 5.2.1 is a direct corollary of the definition of the dynamic programming variables.

5.2.1. Lemma.

$F(t-1) \leq F(s-1)+l_{st}$ *and* $B(s) \leq l_{st}+B(t)$ *for all* $s,t,\ 1 \leq s < t \leq T+1$.

Let s and t be two arbitrary periods ($1 \leq s < t \leq T+1$). The period t is called the *successor* of s, if t immediately succeeds s on the shortest path from s to $T+1$. It follows that $B(s) = l_{st}+B(t)$. We denote the successor of s by $sc(s)$. Analogously, s is called the *predecessor* of t, if s immediately precedes t on the shortest path from 1 to t. Hence, $F(t-1) = F(s-1)+l_{st}$. The predecessor of t is denoted by $pr(t)$.

5.2.2. Lemma.

a) *If* $u = sc(t)$ *and* $t = sc(s)$ *then* $l_{st}+l_{tu} \leq l_{su}$.

b) *If* $s = pr(t)$ *and* $t = pr(u)$ *then* $l_{st}+l_{tu} \leq l_{su}$.

Proof.

a) From $t = sc(s)$ it follows that $B(s) = l_{st}+B(t)$, and $u = sc(t)$ implies $B(t) = l_{tu}+B(u)$. Combining these equations results in $B(s) = l_{st}+l_{tu}+B(u)$. Now the desired inequality follows from $B(s) \leq l_{su}+B(u)$ (Lemma 5.2.1).

b) Analogous to the proof of part a.

□

Lemma 5.2.2 concerns a basic fact of shortests paths, i.e., if (u,v) and (v,w) are two consecutive arcs of a shortest path, then the arc (u,w) has higher or equal costs.

5.2.3. Lemma.

a) *If* $t = sc(s)$, *then* $F(s-1)+B(s) \geq F(t-1)+B(t)$;

b) *If* $s = pr(t)$, *then* $F(s-1)+B(s) \leq F(t-1)+B(t)$.

Proof.

a) $F(s-1)+B(s) = F(s-1)+l_{st}+B(t) \geq F(t-1)+B(t)$, where the inequality follows

from Lemma 5.2.1.

b) Analogous to the proof of part a.

□

Convention

We assume that if $t = sc(s)$, then $d_{s,t-1} > 0$.

This convention excludes degenerate optimal solutions in which period s is determined to be a production period, while actually nothing is produced in that period. It is easy to adapt the algorithms given in the following section such that degenerate solutions are never encountered.

The following lemma is true, under the condition in the convention, although it states a fact which is true, intuitively: if two neighboring production periods are considered, then the latest period has no higher production costs per unit.

5.2.4. Lemma. *If* $t = sc(s)$, *then* $c_s \geq c_t$.

Proof. Let $u = sc(t)$. By Lemma 5.2.2 $l_{st} + l_{tu} \leq l_{su}$, and therefore (after rewriting) $f_t \leq (c_s - c_t) d_{t,u-1}$. Since $f_t \geq 0$, and by our convention $d_{t,u-1} > 0$, it follows that $c_s \geq c_t$.

□

5.2.5. Lemma. *Let* $t = sc(s)$. *Then* $F(s-1) + l_{s\tau} \geq F(t-1) + l_{t\tau}$ *for all* $\tau > sc(t)$.

Proof.

Define $u = sc(t)$.

$F(s-1) + l_{s\tau}$	=	(Definitions of $l_{s\tau}$ and l_{su})
$F(s-1) + l_{su} + c_s d_{u,\tau-1}$	≥	(Lemma 5.2.4)
$F(s-1) + l_{su} + c_t d_{u,\tau-1}$	≥	(Lemma 5.2.2)
$F(s-1) + l_{st} + l_{tu} + c_t d_{u,\tau-1}$	≥	(Lemma 5.2.1)
$F(t-1) + l_{tu} + c_t d_{u,\tau-1}$	=	(Definitions of $l_{t\tau}$ and l_{tu})
$F(t-1) + l_{t\tau}$		

□

5.3. Sensitivity analysis

In this section we give algorithms to calculate, for all the numerical problem parameters, the maximum ranges in which they can vary individually, such that an optimal solution already obtained remains optimal. In most of the algorithms, these (individual) ranges of the problem parameters are calculated simultaneously for all periods. For instance, we will present an algorithm that computes simultaneously, the maximum allowable increases of all the coefficients f_t $(t=1,...,T)$, in $O(T\log T)$ time. We assume that all the relevant information from the forward and backward dynamic programming algorithms is available, i.e., the values $F(t-1)$, $B(t)$ for $t=1,\ldots,T+1$, the periods $sc(t)$ and $pr(t)$ for $t=1,\ldots,T$, the final convex lower envelope associated with the backward recursion, and finally the optimal production schedule.

The parameters are divided into three sets depending on the set of arcs in the dynamic programming network, that change cost if the parameter is altered.

Set I: $f_t, c_t, p_t,\ t=1,\ldots,T$.

If, for a given t, one of these parameters changes, then exactly the arcs that have t as a tail will change in cost. If f_t changes by δ, then all these arcs will also change in cost by δ . If c_t or p_t changes by δ, then the arcs will have a cost change depending on the cumulative demand: the cost of arc (t,τ) will change by $\delta d_{t,\tau-1}$.

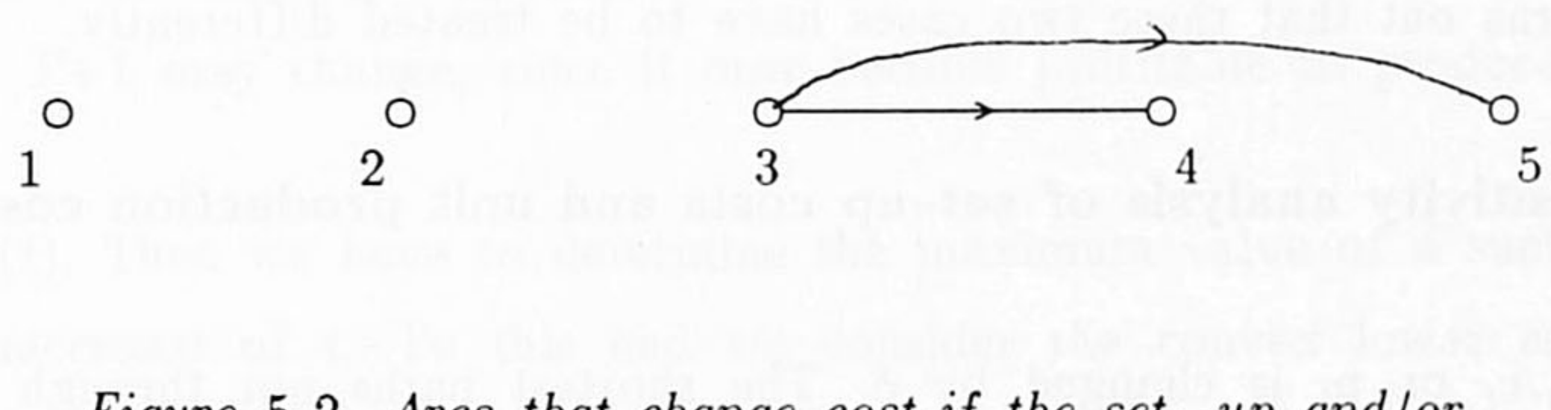


Figure 5.2. Arcs that change cost, if the set-up and/or the production costs of period $t=3$ altered.

Set II: $h_t,\ t=1,\ldots,T$.

If h_t changes by δ, then all c_s, $s\le t$, are perturbed by δ because $c_s=p_s+\sum_{\tau\ge s} h_\tau$. Therefore, the costs of arcs with tail in $\{1,\ldots,t\}$ are changed

by an amount depending on the cumulative demand: $\delta d_{s,u-1}$ for arc (s,u), where $s \leq t$.

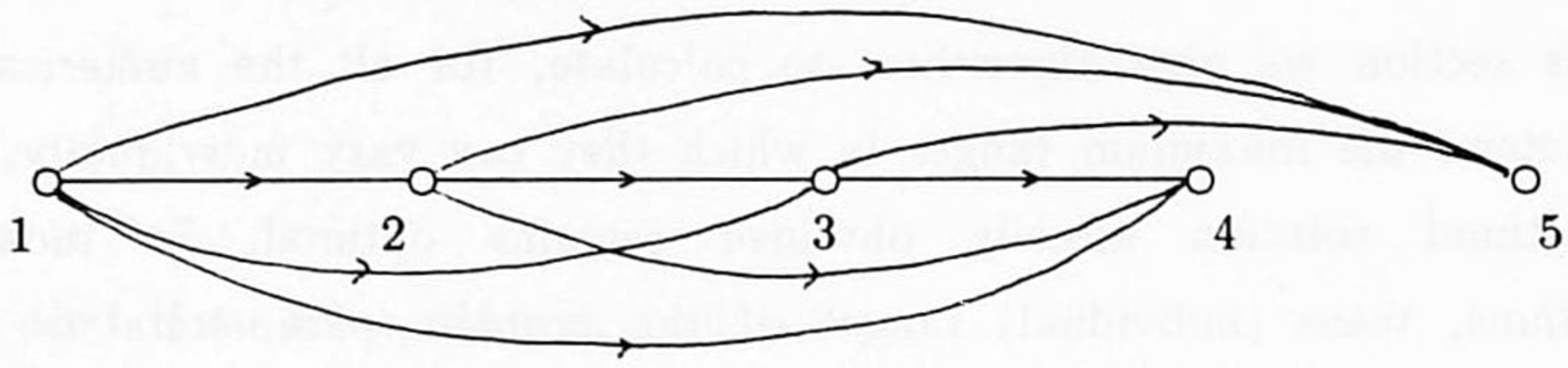


Figure 5.3. Arcs that change cost, if the holding costs of period $t=3$ are altered.

Set III: d_t, $t=1,\ldots,T$.

If d_t is perturbed, then all arc costs in which the demand of period t is involved, will change. These are the arcs (s,u), where $s \leq t$ and $u > t$. The cost change of such an arc is δc_s, where δ is the change in d_t.

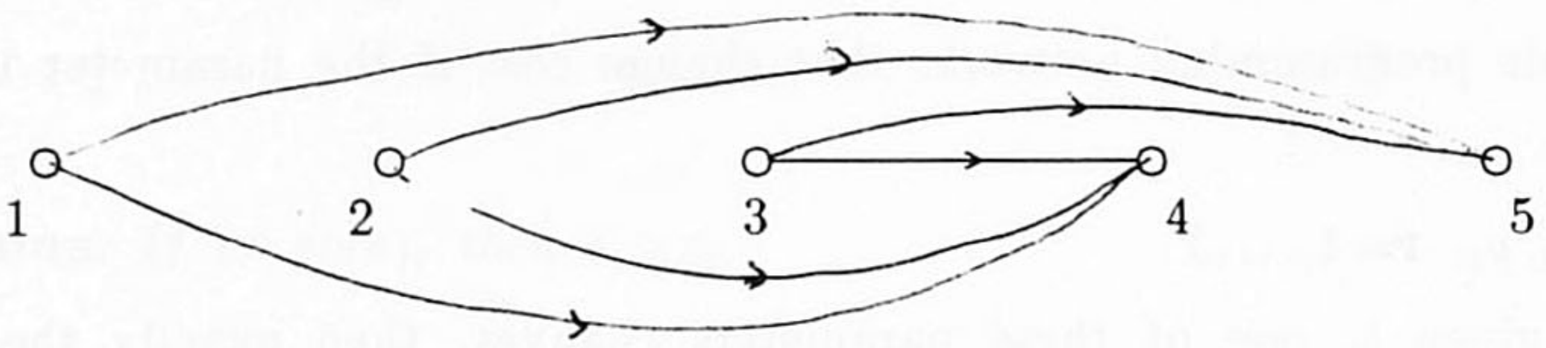


Figure 5.4. Arcs that change cost, if the demand of period $t=3$ is altered.

In the following subsections we treat each of these sets separately. Furthermore, we distinguish between increases and decreases of parameters, since it turns out that these two cases have to be treated differently.

5.3.1. Sensitivity analysis of set-up costs and unit production costs.

Suppose f_t, c_t or p_t is changed by δ. The shortest paths not through t do not have a change in cost. Moreover, the length of the shortest path from 1 to t in the dynamic programming network remains unchanged too, and thus its cost is $F(t-1)$. On the other hand, the costs of all paths from t to $T+1$ do change.

We first consider cost decreases.

Case 1: f_t *decreases to* $f_t-\delta$

The optimal path from 1 to t remains equal, with cost $F(t-1)$ and the optimal path from t to $T+1$ remains equal with cost $B(t)-\delta$.

If t is a production period in the optimal schedule, then this will certainly hold after the cost change. The cost of the optimal schedule is $F(T)-\delta=F(t-1)+B(t)-\delta$. The only upper bound on δ is imposed by the non-negativity of f_t. Thus, δ is bounded by f_t.

If t is not a production period in the optimal schedule, then the shortest path from 1 to $T+1$ does not pass through t. This path has value $B(1)$ and the shortest path through t has value $F(t-1)+B(t)-\delta$. The latter path is shorter if $F(t-1)+B(t)-\delta<B(1)$. Therefore, the optimal path does not change for $\delta\leq F(t-1)+B(t)-B(1)$. Because of the non-negativity of f_t, δ is bounded by $\min\{f_t,F(t-1)+B(t)-B(1)\}$. This shows our first complexity result.

5.3.1. Theorem.

The maximum allowable decrease of f_t *can be calculated in constant time for each* t, $t=1,\ldots,T$.

Case 2: c_t *decreases to* $c_t-\delta$

If t is a production period in the optimal schedule this will remain so, since only paths that contain t have a decrease in cost. However, the shortest path from t to $T+1$ may change, since it may become profitable to produce more in t.

Let $u=sc(t)$. Then we have to determine the maximum value of δ such that u is still the successor of t. To this end we consider the convex lower envelope of the periods $t+1,\ldots,T+1$. See figure 5.5. The successor of t is the period for which the slopes of the line segments incident to it change from smaller than c_t to bigger than c_t. Therefore, as long as that is the case, u remains the successor of t.

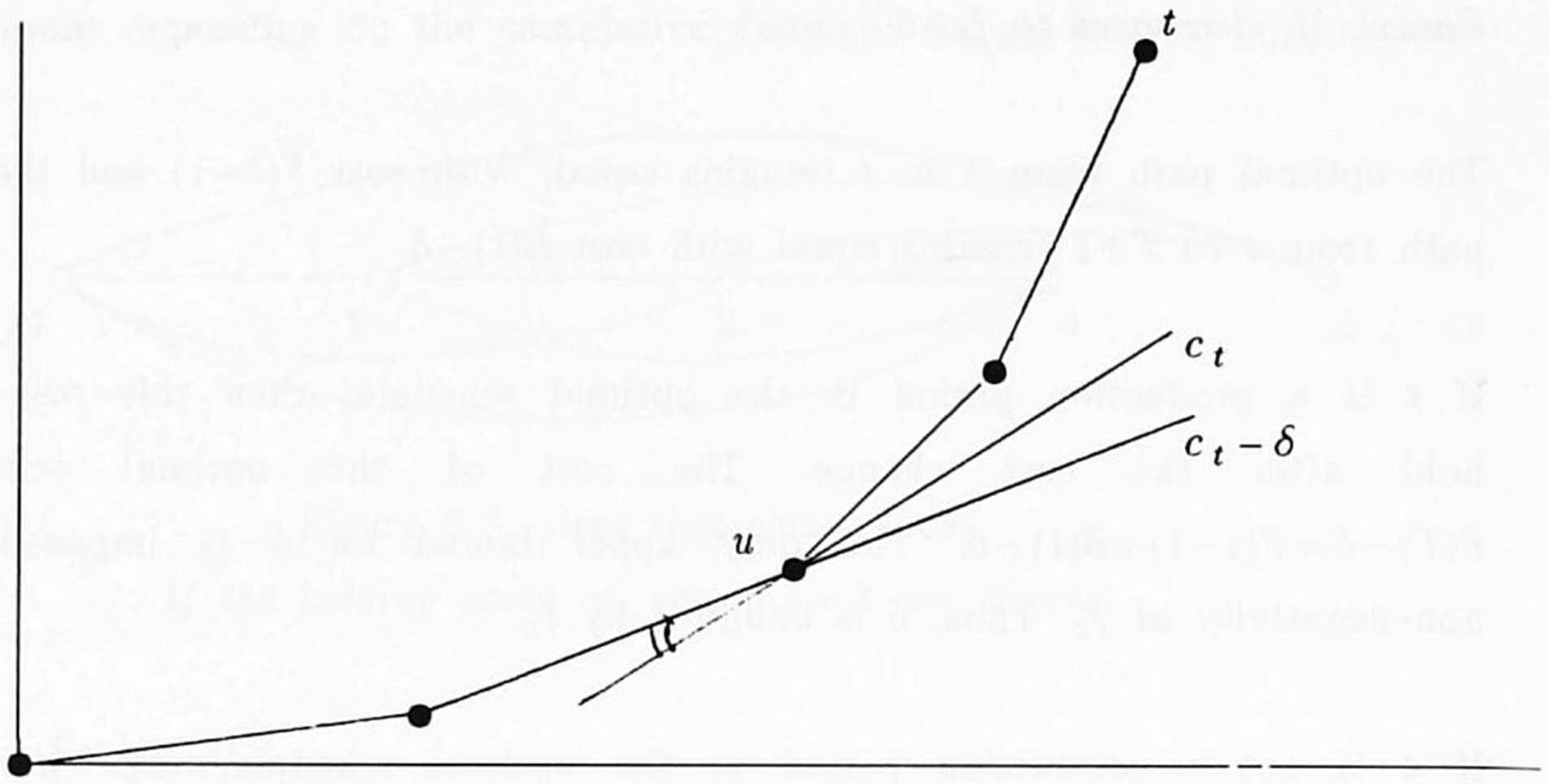


Figure 5.5. *Convex lower envelope of the periods* $t+1,\ldots,T+1$.

Since the unit production costs c_t are decreased, the line tangent to the lower envelope has a decreasing slope. Therefore, we only need to consider the part of the lower envelope consisting of the periods $u,\ldots,T+1$. Clearly, this part of the lower envelope is also available in the final lower envelope, since u is a production period. The maximum decrease of c_t is now easily calculated from the slope of the line segment, immediately left of u. If this slope is denoted by v_u, then u remains a successor of t as long as $c_t-\delta \geq v_u$. Therefore, the optimal schedule remains equal as long as $\delta \leq c_t - v_u$. This value constitutes the upper bound on δ.

We now turn to the case that t is not a production period in the optimal solution. Because the cost of an arc (t,τ) is altered by $\delta d_{t,\tau-1}$, the optimal path from t to $T+1$ has value $f_t+\min_{\tau>t}\{(c_t-\delta)d_{t,\tau-1}+B(\tau)\}$. Period t will not be a production period in an optimal schedule as long as $F(t-1)+f_t+\min_{\tau>t}\{(c_t-\delta)d_{t,\tau-1}+B(\tau)\} > F(T)$. Hence, the maximum allowable decrease of c_t is the value of δ for which

$$\min_{\tau>t}\{(c_t-\delta)d_{t,\tau-1}+B(\tau)\} = F(T)-F(t-1)-f_t$$

Note that the period for which the minimum is attained when δ equals the maximum allowable decrease, is the (possibly new) successor of t, in the

optimal path from 1 to $T+1$, containing t. This successor will be denoted by u. It follows that if δ is equal to the maximum allowable decrease, then both t and this period appear in the *new* final convex lower envelope. We only have to consider the periods $\tau > t$ that appear in the *already known* final convex lower envelope. This follows from the fact that, if c_t is decreased, then the values $B(\tau)$ $(\tau > t)$ do not change. Therefore, the points $(d_{\tau T}, B(\tau))$, $\tau > t$, that do not belong to the known lower envelope can certainly not be present in the new lower envelope, since the latter does not lie above the former.

We now arrive at the actual computation of the maximum allowable decrease of c_t. Consider the final convex lower envelope, restricted to the periods $\tau > t$. In the backward algorithm, we determine the line with given slope that is tangent to this lower envelope; the value of this line in coordinate d_{tT} is the minimum value we are looking for. Now this last value is given, namely $F(T)-F(t-1)-f_t$, and we have to determine the slope of the line that is tangent to the lower envelope and passes through the point $(d_{tT}, F(T)-F(t-1)-f_t)$ (see Figure 5.6). This is easily seen to take $O(\log T)$ time by binary search over the slopes of the line segments in the final lower envelope. Clearly, the slope of this tangent gives us the minimum value of $c_t-\delta$ for which the optimal schedule does not change and thus the maximum value of δ.

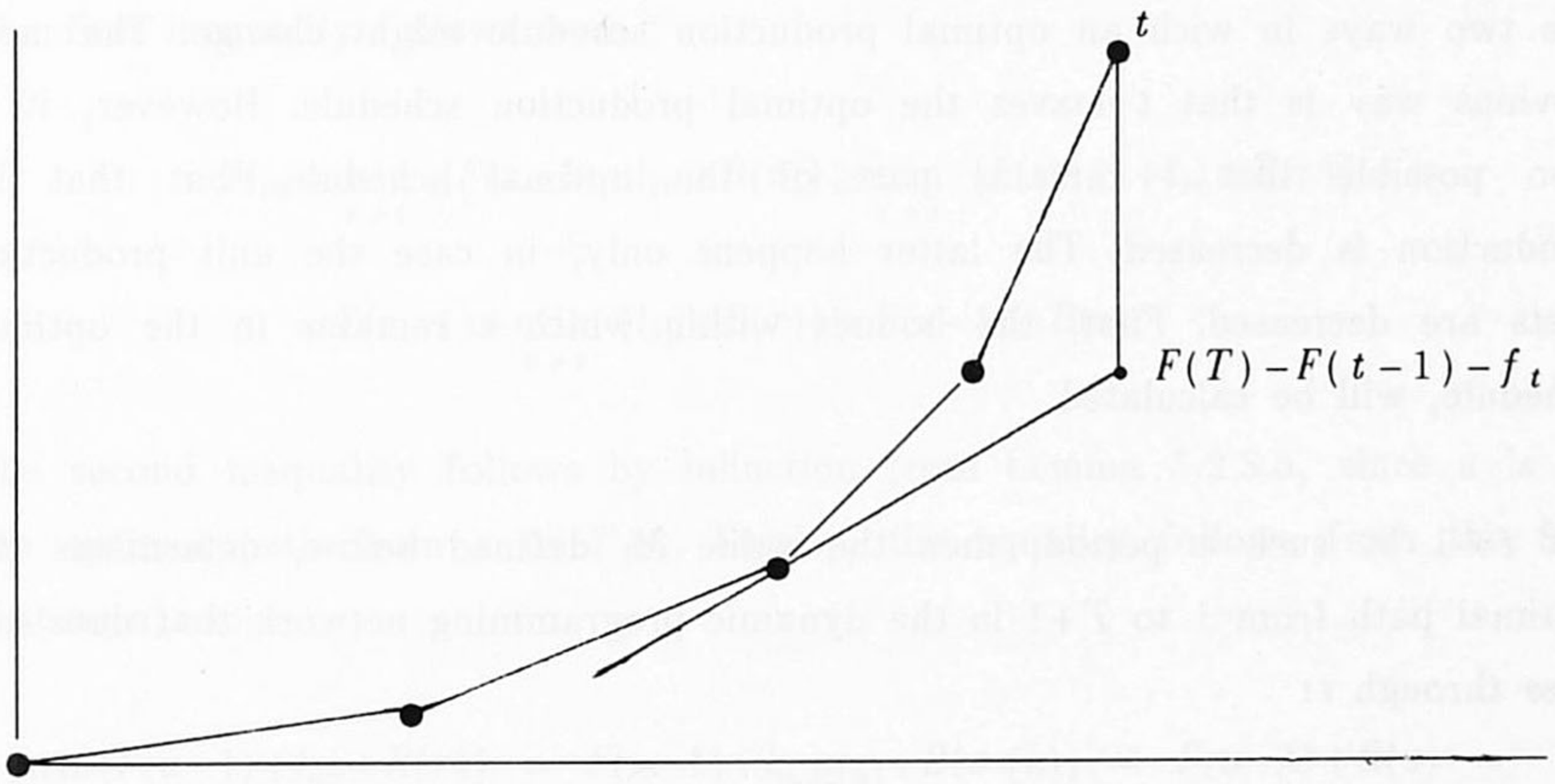


Figure 5.6. Determination of maximum allowable decrease when t is not a production period.

To summarize, we have the following result.

5.3.2. Theorem.

If t is a production period in the optimal schedule, then the maximum allowable decrease of c_t and p_t can be calculated in constant time; in case t is not a production period $O(\log T)$ time suffices.

Note that a change of p_t by δ can be treated in the same manner as a change of c_t by δ. This follows directly from the formula $c_t = p_t + h_{tT}$. However, a change of c_t can also be realized in other ways. For instance, increasing h_t by δ and h_{t-1} by $-\delta$ results in an increase of c_t by δ while leaving the coefficients c_τ, $\tau \neq t$, unaltered.

We will now consider increasing cost coefficients. For a period t that is not a production period in the optimal schedule, the coefficients f_t, c_t and p_t can be increased arbitrarily high, without causing the optimal solution to change. This follows trivially from the fact that $B(t)$ increases, while $B(1)$ remains constant, and therefore $F(t-1)+B(t) \geq B(1)$ continues to hold. Hence, we only have to consider the production periods of the optimal schedule. There are two ways in wich an optimal production schedule might change. The most obvious way is that t leaves the optimal production schedule. However, it is also possible that t remains part of the optimal schedule, but that its production is decreased. The latter happens only, in case the unit production costs are decreased. First, the bounds within which t remains in the optimal schedule, will be calculated.

Let $t \neq 1$ be such a period, then the value M_t defined below, determines the optimal path from 1 to $T+1$ in the dynamic programming network that does not pass through t:

$$M_t \equiv \min_{s < t < \tau} \{F(s-1) + l_{s\tau} + B(\tau)\}$$

Now $M_t = \min_{s<t} M_{st}$, where for $s<t$

$$M_{st} \equiv \min_{\tau > t} \{F(s-1) + l_{s\tau} + B(\tau)\}$$

Before giving algorithms to calculate the maximum allowable increases of f_t and c_t, we will first show how to calculate M_t for all production periods $t \neq 1$ of the optimal schedule simultaneously, in $O(T\log T)$ time. To this end, we partition the periods before t, into two sets:

$$S_t^0 \equiv \{s < t \mid t \text{ is not on the shortest path from } s \text{ to } T+1\}$$

$$S_t^1 \equiv \{s < t \mid t \text{ is on the shortest path from } s \text{ to } T+1\}$$

We define $M_t^0 \equiv \min\{M_{st} \mid s \in S_t^0\}$ and $M_t^1 \equiv \min\{M_{st} \mid s \in S_t^1\}$; clearly, $M_t = \min\{M_t^0, M_t^1\}$. First, we focus on the computation of M_t^0.

5.3.3. Lemma.

Suppose $s \in S_t^0$, *and let* u *be the latest period, smaller than* t, *on the optimal path from* s *to* $T+1$, *i.e.,* $u < t < sc(u)$.

Then $M_{st} \geq F(s-1) + B(s) \geq F(u-1) + B(u) = M_{ut}$.

Proof.

The first inequality follows from

$$M_{st} = \min_{\tau > t} \{F(s-1) + l_{s\tau} + B(\tau)\} \geq \min_{\tau > s} \{F(s-1) + l_{s\tau} + B(\tau)\} =$$

$$= F(s-1) + \min_{\tau > s} \{l_{s\tau} + B(\tau)\} = F(s-1) + B(s)$$

The second inequality follows by induction from Lemma 5.2.3.a, since u is on the optimal path from s to $T+1$. Finally, the equality follows from the fact that $sc(u) > t$:

$$\min_{\tau > t}\{F(u-1) + l_{u\tau} + B(\tau)\} = F(u-1) + l_{u,sc(u)} + B(sc(u)) = F(u-1) + B(u) =$$

$$= \min_{\tau > u}\{F(u-1) + l_{u\tau} + B(\tau)\} \leq \min_{\tau > t}\{F(u-1) + l_{u\tau} + B(\tau)\}$$

Thus

$$M_{ut} = \min_{\tau > t}\{F(u-1)+l_{u\tau}+B(\tau)\} = F(u-1)+B(u)$$

□

From lemma 5.3.3, it follows that calculating M_t^0 is equivalent to determining the minimum of $\{F(s-1)+B(s) | s<t \text{ and } sc(s)>t\}$. Using this fact, we are able to compute values M_t^0 for all production periods $t \neq 1$ of the optimal schedule simultaneously in $O(T\log T)$ time, by the algorithm given below.

The basic ingredient of the algorithm is a binary tree used as a data structure. The leaves of this tree represent the periods $1,\ldots,T$. An intermediate node represents the periods belonging to its subtree. For any period s, we assign the value $F(s-1)+B(s)$ to the nodes $s+1,\ldots,sc(s)-1$. We do not assign this value to each node separately, but such that for each node $\tau \in \{s+1,\ldots,sc(s)-1\}$, there is a node on the path from the root of the tree to the leave representing τ, to which this value is assigned. This is done by labeling only those nodes in the tree that represent a maximal interval of periods that is contained in $\{s+1,\ldots,sc(s)-1\}$ (see figure 5.7). It is not hard to see that this can be done in $O(\log T)$ time for a fixed s.

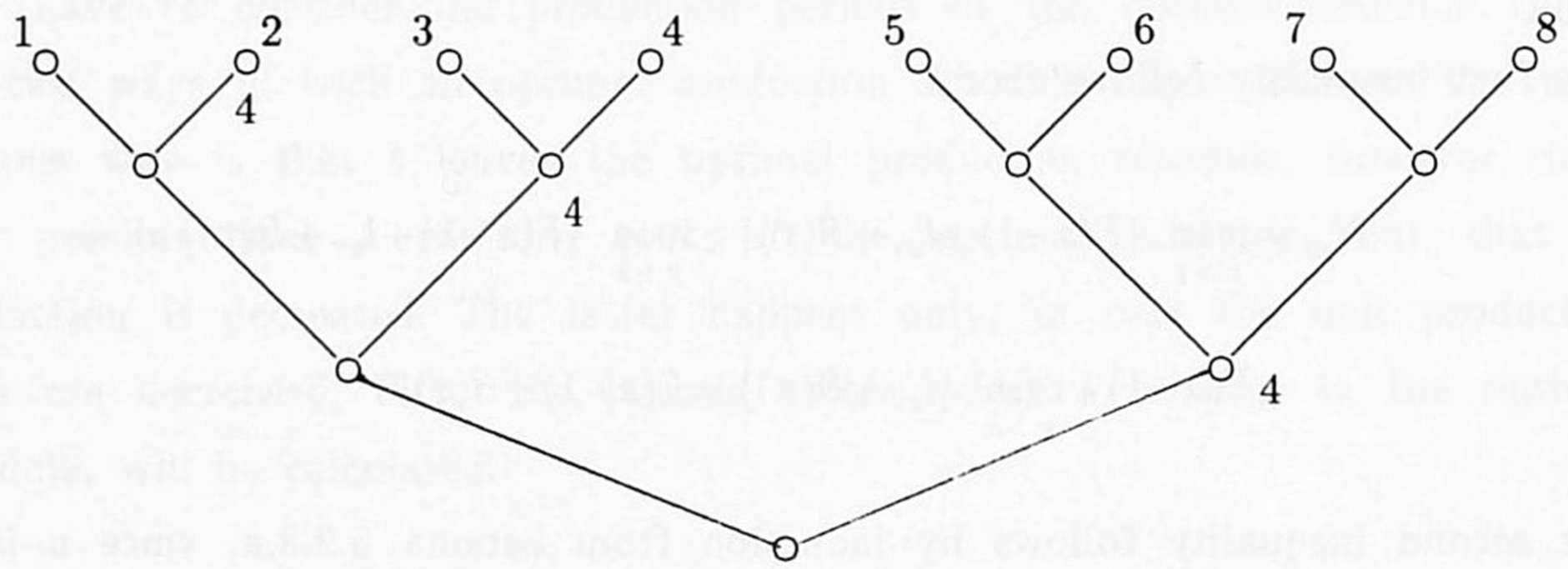


Figure 5.7. *Assigning the value* 4 *to the periods* $\{2,\ldots,8\}$.

Of course, this procedure is repeated for each s in $\{1,\ldots,T\}$. Note that if some node has already been labeled with a value, before the iteration of s, then this label is replaced by $F(s-1)+B(s)$, if and only if $F(s-1)+B(s)$ is smaller than the value of the label. Finally, to find M_t^0 the path from the

leave representing t to the root, is searched for the smallest label, which is M_t^0. This can also be done in $O(\log T)$ time. Since the number of (production) periods is bounded by T, it is easily seen that the whole process takes $O(T\log T)$ time.

We now come to the determination of the values M_t^1 for all production periods $t>1$. The following lemma states a result similar to lemma 5.3.3.

5.3.4. Lemma. *Let* $s\in S_t^1$ *and* $u=sc(s)<t$, *then* $M_{st}\geq M_{ut}$.

Proof.

From Lemma 5.2.5, it follows that

$$F(s-1)+l_{s\tau}+B(\tau)\geq F(u-1)+l_{u\tau}+B(\tau) \qquad \text{for all } \tau>sc(u)$$

Now $s\in S_t^1$ implies that $u\in S_t^1$, and therefore $sc(u)\leq t$. Hence, this inequality certainly holds for all $\tau>t$. Moreover, by definition we have $F(u-1)+l_{u\tau}+B(\tau)\geq M_{ut}$ (for all $\tau>t$). Combining the inequalities gives $F(s-1)+l_{s\tau}+B(\tau)\geq M_{ut}$ (for all $\tau>t$). Therefore, $M_{st}=\min_{\tau>t}\{F(s-1)+l_{s\tau}+B(\tau)\}$, which is at least equal to M_{ut}.

□

From lemma 5.3.4, it follows that to calculate M_t^1, it is sufficient to determine the minimum of $\{M_{st}|sc(s)=t\}$. Consider a period s with $sc(s)=t$. By definition, $M_{st}=F(s-1)+\min_{\tau>t}\{l_{s\tau}+B(\tau)\}$. To evaluate the minimum in this expression, it suffices to determine the line with slope c_s, that is tangent to the convex lower envelope of the points $(d_{\tau T},B(\tau))$ for $\tau>t$. Hence, M_{st} can be calculated in $O(\log T)$ time, given the convex lower envelope for the periods $t+1,\ldots,T+1$. M_t^1 can be determined in $O(m_t\log T)$ time, where $m_t=|\{s|sc(s)=t\}|$. Since $\sum_{t=1}^{T} m_t\leq T$, computing M_t^1 for all relevant periods, takes $O(T\log T)$ time.

In general, we do not have the convex lower envelope of the points $(d_{\tau T},B(\tau))$ $(\tau>t)$, immediately available for all production periods t. However, we can just construct these lower envelopes for decreasing t by the method given in section 3.2. As we have seen, this takes overall $O(T\log T)$ time. Moreover,

determining the set of periods s with $sc(s)=t$ can be done in $O(T)$ time simultaneously, since $sc(s)$ is given for all $s=1,\ldots,T$.

Finally we set $M_t:=\min\{M_t^0, M_t^1\}$, which finishes the whole process of calculating the value of the shortest path from 1 to $T+1$ not containing t.

The calculations above do not concern $t=1$. If $d_1>0$, then period 1 is always a production period and we define $M_1\equiv\infty$. If $d_1=0$ and period 1 is a production period in the optimal schedule, then we define $M_1\equiv B(2)$, which is the value of an optimal schedule in which period 1 is not a production period.

We now proceed with the calculations of the maximum allowable increases of f_t and c_t for a production period t.

Case 3: f_t *increases to* $f_t+\delta$

If t a is production period in the optimal schedule, then the shortest path through t from 1 to $T+1$ has value $F(t-1)+B(t)+\delta$. The corresponding schedule will remain optimal as long as there is no better schedule without t as production period, i.e., as long as $F(t-1)+B(t)+\delta\leq M_t$. So the maximum allowable increase equals $M_t-F(t-1)-B(t)$. Note that $F(t-1)+B(t)=B(1)=F(T)$.

Case 4: c_t *increases* to $c_t+\delta$

Since t is a production period in the optimal schedule, the value of this schedule is now $F(T)+\delta d_{t,sc(t)-1}$. Thus, since M_t is an upper bound on this value, the maximum allowable increase is $\{M_t-F(T)\}/d_{t,sc(t)-1}$, an obviously necesssary condition for the schedule to remain optimal.

However, the optimal schedule can also change, if it becomes more attractive to take an earlier successor of t. Note that arcs (t,τ), $\tau>t$, increase cost by $\delta d_{t,\tau-1}$. To determine the smallest value of δ for which this happens we can use again the convex lower envelope of the points $(d_{\tau T}, B(\tau))$, $\tau>t$. The relevant information is given by the slope of the line segment to the right of the point $(d_{u,T}, B(u))$ in this lower envelope, where u is the successor of t.

Note that the slope v, of this line segment can be obtained at the same time that M_t^1 is determined. It follows that $v-c_t$ is also an upper bound on the maximum allowable increase. The latter value is equal to the minimum of both bounds.

To summarize, we have shown the following.

5.3.5. Theorem.

The maximum allowable increases of f_t, c_t *and* p_t *can be calculated for all* $t=1,\ldots,T$ *simultaneously, in* $O(TlogT)$ *time.*

5.3.2. Sensitivity analysis of the holding costs

Since $c_s=p_s+\sum_{\tau\geq s}h_\tau$, changing h_t to $h_t+\delta$ results in changing c_s by δ for all $s\leq t$. Let $1=t_1<t_2<\ldots.<t_{k-1}\leq t<t_k<\ldots.<t_K<T+1$ denote the production periods of an arbitrary production schedule. Then for this schedule the cost change equals $\delta d_{1,t_k-1}$ (here $t_{K+1}\equiv T+1$). So the value of an optimal schedule is given by

$$V_t(\delta)\equiv \min_{\tau\leq t<u}\{F(\tau-1)+l_{\tau u}+B(u)+\delta d_{1,u-1}\}$$

Case 5: h_t *increases to* $h_t+\delta$

For $1\leq t<u\leq T$ we define $V_{tu}(\delta)\equiv\min_{\tau\leq t}\{F(\tau-1)+l_{\tau u}\}+B(u)+\delta d_{1,u-1}$. By definition, $V_t(\delta)=\min_{u>t}\{V_{tu}(\delta)\}$. Moreover, we can show the following relation.

5.3.6. Lemma.

$min_{u>t}\{V_{tu}(\delta)\}=min_{u>t}\{F(u-1)+B(u)+\delta d_{1,u-1}\}$ *for* $t=1,\ldots,T-1$.

Proof.

For any $u>t$, clearly

$$\min_{\tau\leq t}\{F(\tau-1)+l_{\tau u}\}\geq\min_{\tau<u}\{F(\tau-1)+l_{\tau u}\}=F(u-1) \qquad (5.3.1)$$

and therefore

$$V_{tu}(\delta) \geq F(u-1)+B(u)+\delta d_{1,u-1} \qquad (5.3.2)$$

First suppose $pr(u) \leq t$. Then, using (5.3.1), we obtain

$$F(u-1)=F(pr(u)-1)+l_{pr(u),u} \geq \min_{\tau \leq t}\{F(\tau-1)+l_{\tau u}\} \geq F(u-1)$$

It follows that $\min_{\tau \leq t}\{F(\tau-1)+l_{\tau u}\}=F(u-1)$, and therefore

$$V_{tu}(\delta)=F(u-1)+B(u)+\delta d_{1,u-1} \text{ if } pr(u) \leq t \qquad (5.3.3)$$

Now suppose $pr(u)>t$. Let s be the first period after $t$on the shortest path from 1 to u such that $pr(s) \leq t$. It follows immediately from 5.3.3 that

$$V_{ts}(\delta)=F(s-1)+B(s)+\delta d_{1,s-1}$$

Furthermore, by repeatedly applying Lemma 5.2.3.b, we deduce that $F(s-1)+B(s) \leq F(u-1)+B(u)$, and from $s<u$ it follows that $d_{1,s-1} \leq d_{1,u-1}$. Since $\delta \geq 0$ we obtain

$$V_{ts}(\delta)=F(s-1)+B(s)+\delta d_{1,s-1} \leq F(u-1)+B(u)+\delta d_{1,u-1}$$

Combining this result with inequality 5.3.2 yields $V_{ts}(\delta) \leq V_{tu}(\delta)$. It follows that, while determining $V_t(\delta)=\min_{u>t} V_{tu}(\delta)$, we can restrict ourselves to the periods u with $pr(u) \leq t$. The desired result now follows from 5.3.3.

□

From lemma 5.3.6, it follows that we can obtain the optimal solution value as a function of $\delta \geq 0$, by constructing the lower envelope of the lines $F(u-1)+B(u)+\delta d_{1,u-1}$ as a function of δ, for all $u>t$ (In fact, this lower envelope contains the information for a complete parametric analysis with respect to h_t). If the line corresponding to the current optimal solution is not present in this lower envelope, then the maximum allowable increase of h_t is 0; otherwise, it is equal to the x-coordinate of the first positive

breakpoint of the lower envelope. To obtain the maximum allowable increases for all h_t, $t=1,\ldots,T$, we construct the lower envelopes for decreasing t. Given the convex lower envelope for a fixed t, the lower envelope for $t-1$ is obtained after adding the line $F(t-1)+B(t)+\delta d_{1,t-1}$ to the lower envelope. This means that the lines are added in order of non-increasing slope $d_{1,t-1}$. It is not difficult to see that in this case $O(T)$ time is required to construct all lower envelopes, regarding the techniques discussed in section 3.2. Moreover, it follows that the maximum allowable increases of the parameters h_t are monotone non-decreasing for increasing t. We summarize our main result here.

5.3.7. Theorem.

The maximum allowable increases of all h_t, $t=1,\ldots,T$, *can be calculated simultaneously, in* $O(T)$ *time.*

Case 6: h_t *decreases to* $h_t-\delta$

For this case we do not have a dominance relation similar to lemma 5.3.6. Therefore, we will describe a simple $O(T^2)$ algorithm that can also be used to perform a complete parametric analysis for $\delta \geq 0$. The functions of interest are

$$v_t(\delta) \equiv \min_{\tau \leq t < u} \{F(\tau-1)+l_{\tau u}+B(u)-\delta d_{1,u-1}\} \quad , \quad t=1,\ldots,T$$

For $u>t$ we define $w_{tu} \equiv \min_{\tau \leq t}\{F(\tau-1)+l_{\tau u}\}$ and $v_{tu}(\delta) \equiv w_{tu}+B(u)-\delta d_{1,u-1}$. We first determine the values w_{tu} for all t,u with $1 \leq t < u \leq T+1$. For fixed u $(u=2,\ldots T)$, we compute the values w_{tu}, $1 \leq t < u$, using $w_{1u}=l_{1u}$, and the simple recursion

$$w_{tu} := \min\{w_{t-1,u}, F(t-1)+l_{tu}\} \text{ for } 2 \leq t < u$$

Hence, the determination of all values w_{tu}, $1 \leq t < u \leq T+1$, takes $O(T^2)$ time. Now consider a fixed t $(t=1,\ldots,T)$. Since now the lines $v_{tu}(\delta)$ $(u>t)$, are known and their slopes are already ordered, $v_t(\delta)$ can be determined as the lower envelope of these lines in $O(T)$ time (see section 3.2). It follows that the construction of $v_t(\delta)$ for all $t=1,\ldots,T$ can be done in $O(T^2)$ time and therefore the following holds.

5.3.8. Theorem.

The maximum allowable decreases of all h_t, $t=1,\ldots,T$, *can be calculated simultaneously, in* $O(T^2)$ *time.*

5.3.3. Sensitivity analysis of the demands.

The analysis of changes in the demands resembles the preceding sensitivity analysis of the holding costs. Therefore, we will sometimes refer to subsection 5.3.2 for the details.

If the demand d_t, $t=1,\ldots,T$, changes by δ, then the cost of an arbitrary schedule changes as follows: let $1=t_1<t_2<\ldots.<t_k\leq t<t_{k+1}<\ldots.<t_K<T+1$ denote the production periods of the schedule, then for this schedule the cost change equals δc_{t_k}. So the value of an optimal schedule is given by

$$W_t(\delta)\equiv \min_{s\leq t<\tau}\{F(s-1)+l_{s\tau}+B(\tau)+\delta c_s\}$$

Case 7: d_t increases to $d_t+\delta$

For $s\leq t$ we define $W_{ts}(\delta)\equiv\min_{\tau>t}\{F(s-1)+l_{s\tau}+B(\tau)+\delta c_s\}$. By definition $W_t(\delta)=\min_{s\leq t} W_{ts}(\delta)$. Moreover, we can show the following relation.

5.3.9. Lemma. $min_{s\leq t} W_{ts}(\delta)=min_{s\leq t}\{F(s-1)+B(s)+\delta c_s\}$ *for* $t=1,\ldots,T$.

Proof.

The idea of the proof is analogous to the proof of Lemma 5.3.6. Let $s\leq t$. If $sc(s)>t$, then it is easy to show that $W_{ts}(\delta)=F(s-1)+B(s)+\delta c_s$. If $sc(s)\leq t$, then the following implies that we do not have to consider s.

$$\begin{aligned}
W_{ts}(\delta) &= F(s-1)+\delta c_s+\min_{\tau>t}\{l_{s\tau}+B(\tau)\}\geq && \text{(definition of } B(s)\text{)}\\
&\geq F(s-1)+\delta c_s+B(s)\geq && \text{(Lemma 5.2.3.a)}\\
&\geq F(sc(s)-1)+\delta c_s+B(sc(s))\geq && \text{(Lemma 5.2.4)}\\
&\geq F(sc(s)-1)+\delta c_{sc(s)}+B(sc(s))
\end{aligned}$$

□

From Lemma 5.3.9 it follows that to determine the maximum allowable increase of the parameter d_t, it suffices to determine the first breakpoint of the lower envelope of the lines $F(s-1)+B(s)+\delta c_s$ for all $s \leq t$. Because there is in general no natural order of these lines with respect to their slopes or constant terms, our result here is a factor $\log T$ worse than the comparable result in theorem 5.3.7.

5.3.10. Theorem.

The maximum allowable increases of all d_t, $t=1,\ldots,T$, can be calculated simultaneously, in $O(T\log T)$ time.

Case 8: d_t *decreases to* $d_t-\delta$

Because no equivalent of lemma 5.3.9 is known for this case, we will describe a simple $O(T^2)$ algorithm to perform a complete parametric analysis for $0 \leq \delta \leq d_t$. The functions of interest are

$$w_t(\delta) \equiv \min_{s \leq t < \tau} \{F(s-1)+l_{s\tau}+B(\tau)-\delta c_s\} \quad , \quad r=1,\ldots,T$$

By definition, $w_t(\delta)$ is the lower envelope of the lines $w_{ts}(\delta)$, $s \leq t$, where $w_{ts}(\delta) \equiv F(s-1)+f_s-\delta c_s+\min_{\tau>t}\{c_s d_{s,\tau-1}+B(\tau)\}$. Using a simple recursion we can again calculate the values $\min_{\tau>t}\{c_s d_{s,\tau-1}+B(\tau)\}$ for all $1 \leq s \leq t \leq T+1$ in $O(T^2)$ time. Furthermore, it takes $O(T\log T)$ time to order the coefficients c_s, $s=1,\ldots,T$. Hence, we can compute all relevant lines $w_{ts}(\delta)$, and order them according to non-increasing slope, in $O(T^2)$ time. Subsequently, it takes $O(T)$ time to construct $w_t(\delta)$ for a fixed period t. The discussion above implies our last complexity result.

5.3.11. Theorem.

The maximum allowable decreases of all d_t, $t=1,\ldots,T$, *can be calculated simultaneously, in* $O(T^2)$ *time.*

5.4. Concluding remarks.

In Table 5.1, we have summarized the complexity of our algorithms. The running times refer to the computation of the allowable changes for all similar parameters. We have indicated when a single parameter can be treated separately, in which case the complexity should be divided by T.

parameter	increase	decrease
f_t	$T\log T$	$T^{(*)}$
c_t p_t	$T\log T$	$T\log T^{(*)}$
h_t	T	T^2
d_t	$T\log T$	T^2

Table 5.1. *Summary of complexity results*
(*): *the computations can be carried out for each period separately.*

From the table we see that our algorithms to compute maximum allowable increases and decreases have, for most coefficients, different complexities. The difference for the holding costs is especially striking. To some extent this is not surprising. Such asymmetrical phenomena are also encountered, when discussing sensitivity analysis results for shortest paths and minimum spanning trees, for example. See Wagelmans [51].

We conclude with some open problems. The geometric techniques used to solve ELS have been generalized to solve other lot sizing problems. It would be interesting to study sensitivity analysis of these more general problems. It is well-known from empirical experiments that optimal solutions to ELS are less robust, than solutions produced by heuristic methods, with respect to changes in parameters. It seems worthwhile to verify this analytically, at the same time trying to identify properties that should be present in heuristics that produce solutions which are both good (in a worst case sense) and robust.

Chapter 6

POLYHEDRAL CHARACTERIZATION OF THE ECONOMIC LOT SIZING PROBLEM WITH START-UP COSTS

6.1. Introduction

The subject of chapter 3, was to derive efficient algorithms that solve the economic lot sizing problem (ELS), and the two extensions with start-up costs (ELSS), and backlogging facilities (ELSB). In this chapter we concentrate on the linear description or polyhedral characterization of one of these problems, namely ELSS. The results concerning the linear description of ELS, can be found in Barany et al. [3] and [4]. These results are also described in chapter 1, where an introduction to the aspects of polyhedral theory, used in this chapter, is given. Related results for ELSB can be found in Pochet [38], and Pochet and Wolsey [39].

In [3] and [4], a set of valid inequalities is derived for ELS, the so-called (l,S)-inequalities. It is also proved there, that these inequalities, together with some of the model constraints, suffice to describe the convex hull of ELS. Moreover, it is shown that a large subclass of the (l,S)-inequalities define facets of the convex hull of ELS.

Research on the polyhedral structure of ELSS was initiated by Wolsey [57]. He derived a partial description of ELSS by generalizing the (l,S)-inequalities for ELS to ELSS. The main result of this chapter is to enlarge the set of inequalities in [57], to the so-called (l,R,S)-inequalities, to get a complete linear description of ELSS. It is also shown that most of these inequalities are facet-defining. Moreover, we derive a separation algorithm for these inequalities.

A disadvantage of the polyhedral descriptions of ELS and ELSS is that they contain an exponential number of inequalities. For ELSB we do not even know all facets that describe its convex hull, explicitly. These situations do not occur for the plant location reformulations ELS-UFL, ELSS-UFL and ELSB-UFL for these problems. For ELS-UFL and ELSB-UFL, the given models are strong: optimal solutions to the given LP-relaxations of ELS-UFL and ELSB-UFL will always have integer values for the set-up variables. With respect to ELSS-UFL, things are a little more complicated, since a few more inequalities, the (r,s,τ)-inequalities are necessary to derive a similar result. These inequalities can be found in Wolsey [57], also.

This chapter contains the subjects mentioned above in the following order. In section 6.2, the (l,R,S)-inequalities are introduced. The validity of the (l,R,S)-inequalities is derived with the aid of a multi-commodity flow formulation of ELSS, see Rardin and Wolsey [41]. The so-called di-cut inequalities for this multi-commodity flow problem, described in [41], are shown to imply the (l,R,S)-inequalities. In section 6.3, the polyhedral results with respect to the (l,R,S)-inequalities are given. It is shown that the (l,R,S)-inequalities induce a complete linear description of the convex hull of solutions of ELSS. Moreover, the conditions under which they are facet-defining, are given, together with a proof. The description of a separation algorithm ends section 6.3. The multi-commodity flow formulation is closely related to the uncapacitated facility location formulation ELSS-UFL, since both formulations make use of the disaggregated production variables. The subject of section 6.4 is to show that the (r,s,τ)-inequalities imply the di-cut inequalities, thereby proving that they also imply the (l,R,S)-inequalities. Moreover, it is shown that a subset of these inequalities, the (r,s,s)-inequalities suffice, under mild conditions with respect to the demands.

6.2. The (l,R,S)-inequalities for ELSS

In this section the (l,R,S)-inequalities are introduced. Recall the following mixed integer programming formulation of ELSS. See also section 1.2.

$$\text{(ELSS)} \qquad \min \sum_{t=1}^{T} (g_t z_t + f_t y_t + c_t x_t) \tag{6.2.1}$$

$$\text{s.t.} \quad \sum_{\tau=1}^{T} x_\tau = d_{1T} \tag{6.2.2}$$

$$\sum_{\tau=1}^{t} x_\tau \geq d_{1t} \qquad (1 \leq t \leq T-1) \tag{6.2.3}$$

$$x_t \leq d_{tT} y_t \qquad (1 \leq t \leq T) \tag{6.2.4}$$

$$y_t \leq y_{t-1} + z_t \qquad (y_0 := 0) \qquad (1 \leq t \leq T) \tag{6.2.5}$$

$$y_t, z_t \in \{0,1\} \qquad (1 \leq t \leq T) \tag{6.2.6}$$

$$x_t \geq 0 \qquad (1 \leq t \leq T) \tag{6.2.7}$$

An important structural property of the fixed cost variables in ELSS is the following. If $y_t = 1$ for some $t \in \{1, \ldots, T\}$, then there is a period $s \leq t$ such that $z_s = 1; y_s = \ldots = y_t = 1$. This follows by inductively applying 6.2.5. It leads to the following simple but useful lemma.

6.2.1. Lemma.

Suppose $y_t = 1$, for some t, in a feasible production plan. For any $s < t$, at least one of the following variables $\{y_s, z_{s+1}, z_{s+2}, \ldots, z_t\}$ has value one.

Proof. Let $\tau \leq t$ be as above such that $z_\tau = 1, y_\tau = \ldots = y_t = 1$. If $\tau \leq s$, then $y_s = 1$, and if $\tau > s$, then $z_\tau = 1$. The claim follows.

□

The remainder of this section is devoted to the description of the (l,R,S)-inequalities and a proof of their validity. Take an arbitrary period $l \leq T$, and let $N_l = \{1, \ldots, l\}$. Take $S \subseteq N_l$ and $R \subseteq S$, such that the first element in S is also in R. We define the corresponding (l,R,S)-inequality as follows.

$$\sum_{t \in N_l \setminus S} x_t + \sum_{t \in R} d_{tl} y_t + \sum_{t \in S \setminus R} d_{tl} (z_{p(t)+1} + \ldots + z_t) \geq d_{1l} \tag{6.2.8}$$

where $p(t) = \min\{\tau \mid 0 \leq \tau < t \text{ and } S \cap \{\tau+1, \ldots, t-1\} = \emptyset\}$. Thus, $p(t)$ is the last period in S that precedes t. If none exists, then it is defined zero.

EXAMPLE: $l=9;\ R=\{3\};\ S=\{3,5,8\}$.

The coefficients of the inequality are given in the following table.

t	1	2	3	4	5	6	7	8	9
z_t				d_{59}	d_{59}	d_{89}	d_{89}	d_{89}	
y_t			d_{39}						
x_t	1	1		1		1	1		1
$p(t)$	0	0	0	3	3	5	5	5	8

It should be noted that the inequalities derived in Wolsey [57], are a special case of the (l,R,S)-inequalities. Those inequalities have the property that for each period $t\in S\backslash R$, the preceding period $t-1\in S$. Therefore, the example above is not included, because the periods 5 and 8 are in S, and the periods 4, 6 and 7 are in $N_l\backslash S$.

In the remainder of this section, the validity of the (l,R,S)-inequalities is established. In Rardin and Wolsey [41], it has been shown that these inequalities can be derived from the di-cut inequalities in a suitable uncapacitated fixed charge network flow model. We will follow their line of proof.

The flow network for ELSS consists of the following vertices and arcs. There is a source s, and there are three layers of nodes: $\{u_t|1\le t\le T\}$, $\{v_t|1\le t\le T\}$, and $\{w_t|1\le t\le T\}$. The arcs (s,u_t), $(1\le t\le T)$, model the fixed start-up charges, i.e., if such an arc contains a positive flow, then $z_t=1$. The arcs (u_t,v_t), $(1\le t\le T)$, model the fixed set-up charges. The arcs (v_t,u_{t+1}), $(1\le t\le T-1)$, are included to allow for multiple set-ups in consecutive periods without a start-up in these periods. The arcs (v_t,w_t), $(1\le t\le T)$, model the production in period t. These are not fixed charge arcs, but the production is exactly equal to the flow through the arc. Finally, the flows through the arcs (w_t,w_{t+1}), $(1\le t\le T-1)$, denote the inventory at the end of period t, or equivalently, at the beginning of period $t+1$. There are T different commodities τ, $(1\le\tau\le T)$, in the network, consisting of the source s and sink w_τ, and with a demand of d_τ

units of flow. For a commodity τ, the flow through the arcs (v_t, w_t), $(1 \leq t \leq \tau)$, is the production in period t for period τ, and therefore, it is denoted by $x_{t\tau}$. The arcs (w_t, w_{t+1}), $(1 \leq t \leq \tau - 1)$, contain the inventory at the end of period t for demand in period τ. It is denoted by $I_{t\tau}$.

EXAMPLE:

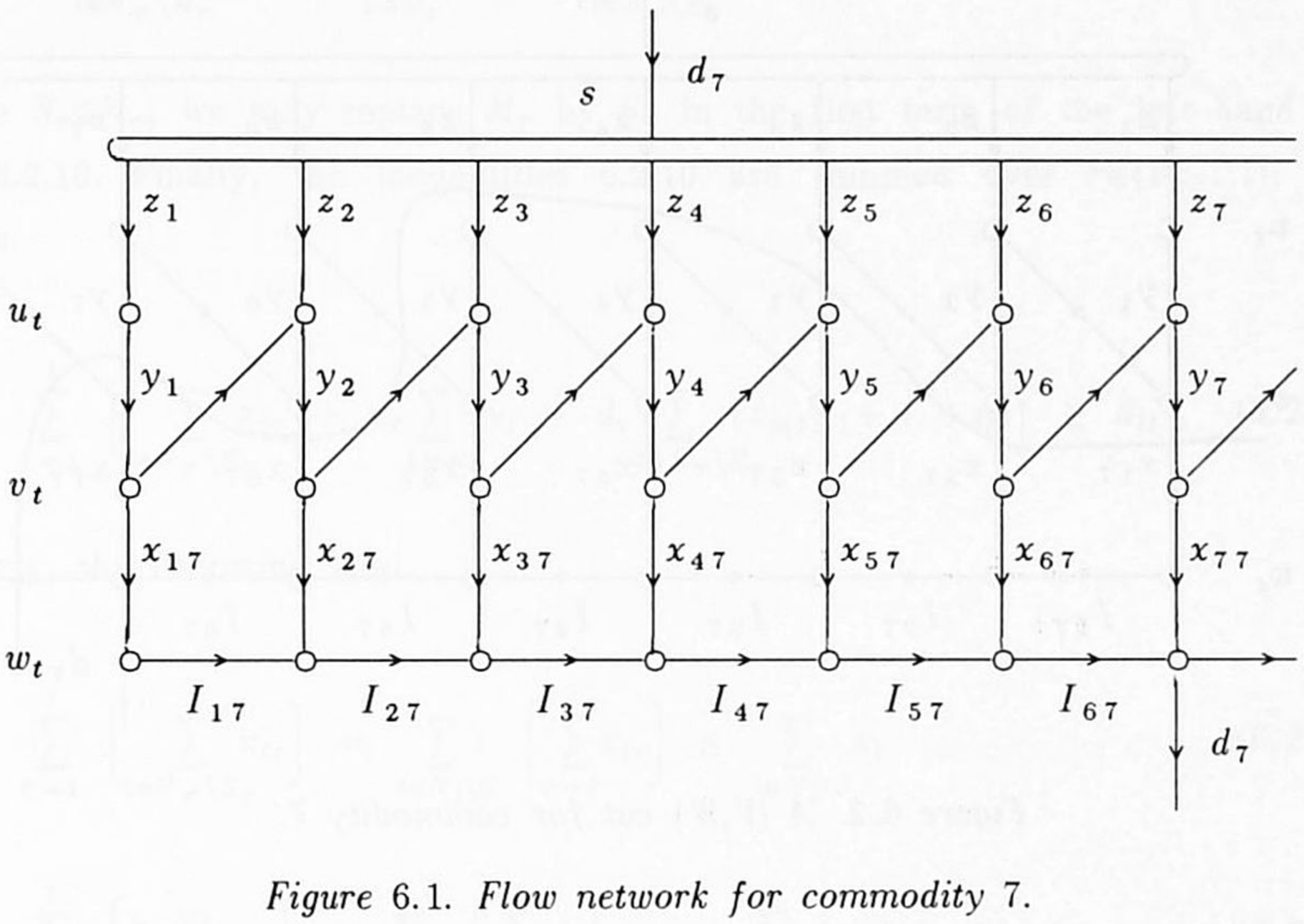


Figure 6.1. *Flow network for commodity* 7.

Consider a feasible flow of d_τ units of a given commodity τ. The flow "through" a cut (V,W) separating s and w_τ is at least d_τ units. For each type of arc one can derive an upperbound on the flow through such arcs as follows. The flow through an arc (s, u_t) is bounded by $d_\tau z_t$, the flow through an arc (u_t, v_t) is bounded by $d_\tau y_t$. Finally, the flow through an arc (v_t, w_t) equals $x_{t\tau}$. We will consider cuts that "contain" these types of arcs only. In that case, it follows trivially that the sum of these upper bounds for all arcs in the cut, constitutes an upperbound on the flow d_τ. The cuts (V,W) that we consider for a given period τ, have the following properties.

Let $s \in V$. For each period $t \in \{\tau+1, \ldots, T\}$: $u_t \in V$, $v_t \in V$, and $w_t \in V$;
For each period $t \in \{1, \ldots, \tau\}$: if $u_t \in W$, then $v_{t-1} \in W$;
if $v_t \in V$, then $u_t \in V$;
$w_t \in W$.

It follows immediately from the property: $u_t \in W$ implies $v_{t-1} \in W$, that the arcs (v_t, u_{t-1}) are not in the cut (V,W). Moreover, the arcs (w_t, w_{t+1}) are not in the cut either, because all w_t $(1 \leq t \leq \tau)$ are in W.

EXAMPLE:

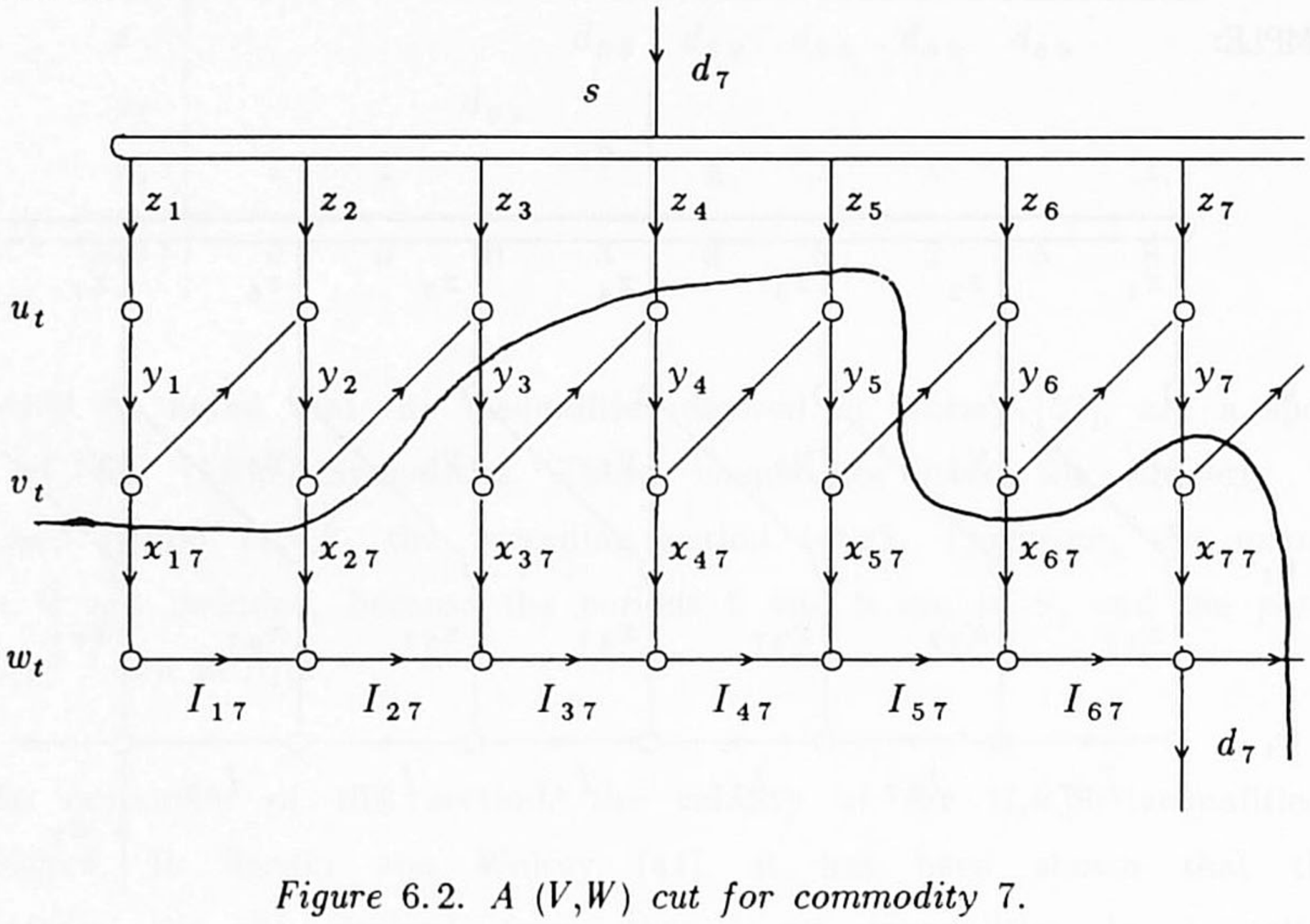


Figure 6.2. *A* (V,W) *cut for commodity* 7.

The periods t for which $u_t \in V$ and $v_t \in W$ are denoted by r_k $(1 \leq k \leq K)$, in increasing order, i.e., $1 \leq r_1 < r_2 < \ldots < r_K \leq \tau$. For each period r_k a period $s_k \geq r_k$ is defined. This period is the maximal period, such that for the periods t in the set $\{r_k+1,\ldots,s_k\}$, the vertex $u_t \in W$. Thus, $s_k < r_{k+1}$ for $k < K$. Defining the union of the batches $\{r_k,\ldots,s_k\}$ $(1 \leq k \leq K)$ by M, we get the following inequality.

$$\sum_{t \in N_\tau \setminus M} x_{t\tau} + d_\tau \sum_{k=1}^{K} y_{r_k} + d_\tau \sum_{k=1}^{K} (z_{r_k+1} + \ldots + z_{s_k}) \geq d_\tau \qquad (6.2.9)$$

This set of inequalities implies the (l,R,S)-inequalities, as will be shown. Take an arbitrary l, and choose an arbitrary $S \subseteq \{1,\ldots,l\}$. Finally, take $R \subseteq S$ such that the first element of S is in R.

For each period $\tau \in \{1,\ldots,l\}$, an inequality of the type 6.2.9 will be derived. In order to do so, we define $S_\tau = S \cap N_\tau$ and $R_\tau = R \cap N_\tau$. The union of the set R_τ with

the periods in $\{p(t)+1,\ldots,t\}$ for all $t\in S_\tau\backslash R_\tau$ is called M_τ. This set M_τ consists of a set of batches of periods $\{r_k,\ldots,s_k\}$ $(1\leq k\leq K)$, such that $r_k\in R_\tau$ and $\{r_k+1,\ldots,s_k\}\cap R_\tau=\emptyset$ (A batch is a set of consecutive periods). Thus the following inequality is of the type 6.2.9.

$$\sum_{t\in N_\tau\backslash M_\tau} x_{t\tau} + d_\tau \sum_{t\in R_\tau} y_t + d_\tau \sum_{t\in S_\tau\backslash R_\tau} (z_{p(t)+1}+\ldots+z_t) \geq d_\tau \qquad (6.2.10)$$

Since $S_\tau\subseteq M_\tau$, we may replace M_τ by S_τ in the first term of the left-hand side of 6.2.10. Finally, the inequalities 6.2.10 are summed over $\tau\in\{1,\ldots,l\}$. This gives

$$\sum_{\tau=1}^{l}\left[\sum_{t\in N_\tau\backslash S_\tau} x_{t\tau} + d_\tau\sum_{t\in R_\tau} y_t + d_\tau \sum_{t\in S_\tau\backslash R_\tau} (z_{p(t)+1}+\ldots+z_t)\right] \geq d_{1l} \qquad (6.2.11)$$

Clearly, the following hold.

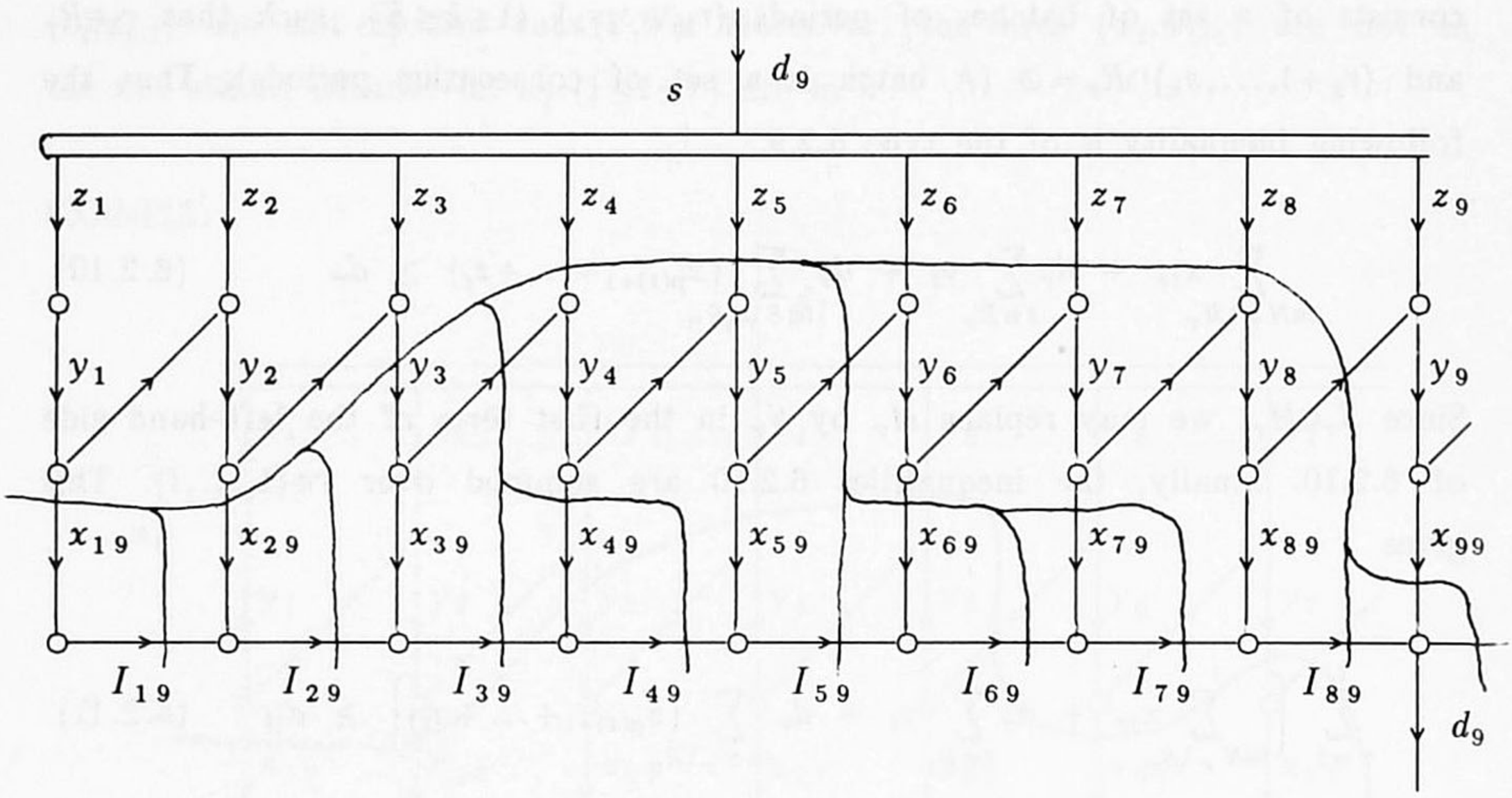
$$\sum_{\tau=1}^{l}\left[\sum_{t\in N_\tau\backslash S_\tau} x_{t\tau}\right] = \sum_{t\in N_l\backslash S}\left[\sum_{\tau=t}^{l} x_{t\tau}\right] \leq \sum_{t\in N_l\backslash S} x_t \qquad (6.2.12)$$

$$\sum_{\tau=1}^{l}\left[d_\tau\sum_{t\in R_\tau} y_t\right] = \sum_{t\in R}\left[\sum_{\tau=t}^{l} d_\tau\right] y_t = \sum_{t\in R} d_{tl}y_t \qquad (6.2.13)$$

$$\sum_{\tau=1}^{l}\left[d_\tau \sum_{t\in S_\tau\backslash R_\tau} (z_{p(t)+1}+\ldots+z_t)\right] = \sum_{t\in S\backslash R} d_{tl}(z_{p(t)+1}+\ldots+z_t) \qquad (6.2.14)$$

Substituting 6.2.12, 6.2.13 and 6.2.14 in the left-hand side of6.2.11, yields the desired (l,R,S)-inequality 6.2.8.

This section is ended with an example. This example is taken from the (l,R,S)-inequality at the beginning of this section.



$$
\begin{array}{llllllll}
x_{11} & & & & & & & \geq d_1 \\
x_{12}+x_{22} & & & & & & & \geq d_2 \\
x_{13}+x_{23}+d_3y_3 & & & & & & & \geq d_3 \\
x_{14}+x_{24}+d_4y_3+x_{44} & & & & & & & \geq d_4 \\
x_{15}+x_{25}+d_5y_3+ & d_5z_4+d_5z_5 & & & & & & \geq d_5 \\
x_{16}+x_{26}+d_6y_3+ & d_6z_4+d_6z_5+x_{66} & & & & & & \geq d_6 \\
x_{17}+x_{27}+d_7y_3+ & d_7z_4+d_7z_5+x_{67}+ & x_{77} & & & & & \geq d_7 \\
x_{18}+x_{28}+d_8y_3+ & d_8z_4+d_8z_5+ & d_8z_6+ & d_8z_7+d_8z_8 & & & & \geq d_8 \\
x_{19}+x_{29}+d_9y_3+ & d_9z_4+d_9z_5+ & d_9z_6+ & d_9z_7+d_9z_8+x_{99} & & & & \geq d_9 \\
\hline
 & d_{59}z_4+d_{59}z_5+ & d_{89}z_6+ & d_{89}z_7+d_{89}z_8+ & & & & + \\
d_{39}y_3+ & & & & & & & \geq d_{19} \\
x_1+x_2+ & x_4+ & x_6+ & x_7+ & x_9 & & &
\end{array}
$$

6.3. The convex hull of ELSS.

6.3.1. Linear description of ELSS with the (l,R,S)-inequalities.

Consider the following sets of constraints.

$$\sum_{t=1}^{T} x_t = d_{1T} \tag{6.3.1}$$

$$z_t \le 1 \qquad (1 \le t \le T) \tag{6.3.2}$$

$$y_t \le 1 \qquad (1 \le t \le T) \tag{6.3.3}$$

$$y_t \le y_{t-1} + z_t \qquad (y_0 := 0) \qquad (1 \le t \le T) \tag{6.3.4}$$

$$x_t \ge 0 \qquad (1 \le t \le T) \tag{6.3.5}$$

$$y_t \ge 0 \qquad (1 \le t \le T) \tag{6.3.6}$$

$$z_t \ge 0 \qquad (1 \le t \le T) \tag{6.3.7}$$

$$\sum_{t \in N_l \setminus S} x_t \; + \; \sum_{t \in R} d_{tl} y_t \; + \; \sum_{t \in S \setminus R} d_{tl}(z_{p(t)+1} + \ldots + z_t) \;\ge\; d_{1l} \tag{6.3.8}$$

For all $l \in \{1,\ldots,T\}$, $S \subseteq N_l$ and $R \subseteq S$, such that the first element of S is in R.

Note that the inequalities 6.2.3 are $(t,\emptyset,\emptyset)$-inequalities. The inequalities 6.2.4 can be derived from 6.3.1 and 6.3.8, where $S = R = \{t\}$ and $l = T$. The remainder of this section will be devoted to proving the following theorem.

6.3.1. Theorem.

The sets of inequalities 6.3.1,...,6.3.8 *describe the convex hull of ELSS.*

The technique we will use to prove the theorem is somewhat different from the usual techniques. Basically, the idea is to show that for an arbitrary objective function, the set of optimal solutions satisfies one of the inequalities 6.3.2,...,6.3.8 as equality. Thus 6.3.2,...,6.3.8 must include all facets of the convex hull of solutions. See section 1.4, for a more detailed description.

We consider an arbitrary cost function $\sum_{t=1}^{T}(\alpha_t x_t + \beta_t y_t + \gamma_t z_t)$, and we denote its set of optimal solutions by $M(\alpha,\beta,\gamma)$.

Case 0: $\min\{\alpha_t|t=1,\ldots,T\}=\delta\neq 0$

As $\sum_{t=1}^{T}x_t=d_{1T}$ we can subtract δ times the inequality 6.3.1 from the objective function without changing the set of optimal solutions.

Thus, in the following we can assume that $\min\{\alpha_t|t=1,\ldots,T\}=0$.

Case 1: $\gamma_t<0$ for some $t\in\{1,\ldots,T\}$.

Any solution with $z_t=0$ can be improved by setting $z_t=1$. Thus $M(\alpha,\beta,\gamma)\subseteq\{(x,y,z)|z_t=1\}$.

Case 2: $\gamma_t\geq 0$ for all t, $\beta_t<0$ for some t.

i) $\beta_t+\gamma_t<0$ for some t.

Any solution with $y_t=0$ can be improved by setting $y_t=z_t=1$. Thus, $M(\alpha,\beta,\gamma)\subseteq\{(x,y,z)|y_t=1\}$.

ii) $\beta_t+\gamma_t\geq 0$ for all t.

Let $s=\min\{t|\beta_t<0\}$. We show that $M(\alpha,\beta,\gamma)\subseteq\{(x,y,z)|y_{s-1}+z_s=y_s\}$. As $\gamma_s>0$, any solution with $y_{s-1}=z_s=1$ can be improved by setting $z_s=0$. In addition, any solution with $y_{s-1}+z_s=1$ and $y_s=0$ can be improved by setting $y_s=1$. Thus, the claim follows.

We are left with objective functions satisfying $\min\{\alpha_t|t=1,\ldots,T\}=0$; $\beta_t\geq 0$ $(t=1,\ldots,T)$; $\gamma_t\geq 0$ $(t=1,\ldots,T)$. In the following, l, S and R are determined, step by step, in this order. First, the period l is fixed. This is done such that the demands of the periods $\{l+1,\ldots,T\}$ can be produced at no cost. For notational convenience, a period $T+1$ is defined with $\alpha_{T+1}=0$, $\beta_{T+1}=0$, $\gamma_{T+1}=0$, and the demand of $T+1$ is supposed to be positive.

Choice of *l*: Take k and m $(k\leq m)$, both minimal, such that $\gamma_k=\beta_k=\ldots=\beta_m=\alpha_m=0$. The period l is the last period in $\{1,\ldots,m-1\}$ with positive demand, i.e., $d_l>0$, and $d_{l+1}=\ldots=d_{m-1}=0$. If $d_{1,m-1}=0$, then $l:=0$.

Case 3: Since $d_{l+1,m-1}=0$, any production in the periods $\{l+1,\ldots,T\}$ can be shifted to period m at no cost, by taking $z_k=y_k=\ldots=y_m=1$. It follows that

i) If $\alpha_t>0$ for some $t\geq l+1$, then $M(\alpha,\beta,\gamma)\subseteq\{(x,y,z)|x_t=0\}$.

ii) If $\beta_t>0$ for some $t\geq l+1$ or $t\geq k$, then $M(\alpha,\beta,\gamma)\subseteq\{(x,y,z)|y_t=0\}$.

iii) If $\gamma_t > 0$ for some $t \geq l+1$ or $t \geq k$, then $M(\alpha,\beta,\gamma) \subseteq \{(x,y,z) | z_t = 0\}$.
If the contrary holds in one of these cases, then the solution can be improved by taking the solution mentioned above.

Note that, if $l=0$, then we are finished. In that case the only objective function left is the one with zero cost coefficients for all variables. Thus, in the following we may assume that $l>0$. Moreover, we may suppose that for $t \geq \min\{k,l+1\}$ we have $\beta_t = \gamma_t = 0$, and for $t \geq l+1$, we have $\alpha_t = 0$. We proceed by specifying the choices of S and R.

Choice of *S*: $S := \{t \leq l | \alpha_t = 0\}$.
Choice of *R*: $R := \{t \in S | \beta_t > 0\}$.

With regard to the following case, it is important to note that, if $l=T$, then S is not empty, since $\min\{\alpha_t | t = 1,\ldots,T\} = 0$.

Case 4: If $S = \emptyset$, then $M(\alpha,\beta,\gamma) \subseteq \{(x,y,z) | \sum_{t=1}^{l} x_t = d_{1l}\}$.
$S = \emptyset$ implies that $\alpha_t > 0$ for $t \in \{1,\ldots,l\}$. Therefore, if $I_l > 0$, then the production costs can be reduced by transferring units of production from the last production period in $\{1,\ldots,l\}$ to $l+1$. Note that $\alpha_{l+1} = \beta_{l+1} = \gamma_{l+1} = 0$. This proves the claim, since $I_l = 0$ implies $\sum_{t=1}^{l} x_t = d_{1l}$.

In the following, we may suppose that $S \neq \emptyset$.

Case 5: Suppose, for some $t \in S \setminus R$, there is an $s \in \{p(t)+1,\ldots,t\}$ with $\beta_s > 0$. Then for this t the following holds. $M(\alpha,\beta,\gamma) \subseteq \{(x,y,z) | y_s + z_{s+1} + \ldots + z_t = y_t\}$.
Note that the equation $y_s + z_{s+1} + \ldots + z_t = y_t$ implies that $y_{\tau-1} + z_\tau = y_\tau$ $(s < \tau \leq t)$. Before we proceed, some properties of the coefficients of the variables are stated. First, $\beta_s > 0$ and $\beta_{s+1},\ldots,\beta_t = 0$ (since $t \notin R$). Thus, we have $s<t$. Second, $\alpha_s,\ldots,\alpha_{t-1} > 0$, since $s \in \{p(t)+1,\ldots,t-1\}$, and $\alpha_t = 0$, since $t \in S$. Finally, if $\gamma_\tau = 0$ for some $\tau \in \{s+1,\ldots,t\}$, then $\gamma_\tau = \beta_\tau = \ldots = \beta_t = \alpha_t = 0$, which implies $l<t$, a contradiction. It follows that $\gamma_{s+1},\ldots,\gamma_t > 0$. Concluding, the variables on the left-hand side of the equation have positive cost coefficients, and the variable on the right-hand side has coefficient zero.
First, suppose that $y_s + z_{s+1} + \ldots + z_t \geq 2$. Let u be the first period in $\{s+1,\ldots,t\}$

such that the variable z_u has value 1. If $y_s=1$, then a cost reduction can be achieved by setting $y_{s+1},\ldots,y_u=1$, and $z_u=0$. If $y_s=0$, then let v be the first period after u with $z_v=1$. In this case, a cost reduction can be obtained by setting $y_u=\ldots=y_v=1$ and $z_v=0$.

Second, suppose that $y_s+z_{s+1}+\ldots+z_t=1$ and $y_t=0$. If no production takes place in $s,..,t-1$, then setting $y_s+z_{s+1}+\ldots+z_t=0$ leads to a cost reduction without losing feasibility. Otherwise, production takes place in one or more of the periods. $\{s,\ldots,t-1\}$. Let u be the last production period in $\{s,\ldots,t-1\}$. If $I_{t-1}>0$, then production from period u can be transferred to t, at a cost reduction, since $\beta_{u+1}=\ldots=\beta_t=\alpha_t=0$, and $\alpha_u,\ldots,\alpha_{t-1}>0$. Thus, we may assume that $I_{t-1}=0$. Then d_t must be zero, since $y_t=0$, which implies $l>t$. Thus, the positive demand $d_{t+1,l}$ is produced at a positive cost in some of the periods in the set $\{t+1,\ldots,l\}$, otherwise l would have been chosen smaller. But now it is cheaper to produce this demand in period t at zero cost. Of course, this implies $y_{u+1}=\ldots=y_t=1$, where u is the last production period in $\{s,\ldots,t-1\}$.

In the following we may assume that for each $t\in S\backslash R$ we have $\beta_{p(t)+1}=\ldots=\beta_t=0$. In the following case we suppose that the first element in S in not in R.

Case 6: If $\beta_1=\ldots=\beta_t=0$, where t is the first element in S, then we have the following: $M(\alpha,\beta,\gamma)\subseteq\{(x,y,z)|\sum_{\tau=1}^{t-1} x_\tau=d_{1,t-1}\}$.

By definition of t we have $\{1,\ldots,t-1\}\cap S=\emptyset$, and thus $\alpha_1,\ldots,\alpha_{t-1}>0$. Moreover, since $t\in S$, $\alpha_t=0$.

If $y_1+\ldots+y_t=0$, then $\sum_{\tau=1}^{t-1} x_\tau=0=d_{1,t-1}$.

If $y_1+\ldots+y_t\geq 1$, then we can produce in t at no additional production costs, and thus, since $\alpha_1,\ldots,\alpha_{t-1}>0$, we have $I_{t-1}=0$, see case 4. Therefore, $\sum_{\tau=1}^{t-1} x_\tau=d_{1,t-1}$.

It follows from case 6, that we may assume that the first period in S is in R.

Case 7: We claim that the following (l,R,S)-inequality

$$\sum_{t\in N_l\backslash S} x_t + \sum_{t\in R} d_{tl}y_t + \sum_{t\in S\backslash R} d_{tl}(z_{p(t)+1}+\ldots+z_t) \geq d_{1l}$$

is satisfied as equality for all points in $M(\alpha,\beta,\gamma)$.

The following properties concerning the cost coefficients are valid.

For $t\in N_l\backslash S$, $\alpha_t>0$, by definition of S.

For $t\in R$, $\beta_t>0$, $\alpha_t=0$, by definition of R.

For $t\in S\backslash R$, $\gamma_{p(t)+1},\ldots,\gamma_t>0$, since $\beta_{p(t)+1}=\ldots=\beta_t=\alpha_t=0$. Otherwise, $l<t$.

i) Suppose $y_t=0$ for all $t\in R$ and $z_{p(t)+1}+\ldots+z_t=0$ for all $t\in S\backslash R$. Then $y_t=0$ for all $t\in S$, which implies that there is no production in the periods of S. Moreover, the contribution of the variables in the left-hand side of the periods in S is zero. It is left to prove that $\sum_{t\in N_l\backslash S} x_t=d_{1l}$. If $I_l>0$, then the costs can be decreased by transferring production from the last production period in $\{1,\ldots,l\}$ to $l+1$. Recall that $\alpha_{l+1}=\beta_{l+1}=\gamma_{l+1}=0$. Thus $I_l=0$. This proves the claim.

Take the minimal $t\in S$, such that $y_t=1$ (if $t\in R$) or $z_{p(t)+1}+\ldots+z_t\geq 1$ (if $t\in S\backslash R$).

ii) If $t\in R$, then by the minimality of t this gives the following.

a) For $\tau<t$, $\tau\in R$, $y_\tau=0$.

b) For $\tau<t$, $\tau\in S\backslash R$, $z_{p(\tau)+1},\ldots,z_\tau=0$.

Since $\alpha_t=0$, and $y_t=1$, the production of d_{tl} can take place at no additional cost in t. Therefore, the following hold.

c) For $\tau>t$, $\tau\in N_l\backslash S$, $x_\tau=0$, since $\alpha_\tau>0$.

d) For $\tau>t$, $\tau\in R$, $y_\tau=0$, since $\beta_\tau>0$.

e) For $\tau>t$, $\tau\in S\backslash R$, $z_{p(\tau)+1}=\ldots=z_\tau=0$, since $\gamma_{p(\tau)+1},\ldots,\gamma_\tau>0$.

Thus, the value of the left-hand side of the (l,R,S)-inequality is reduced to the quantity $\sum_{\tau\in N_{t-1}\backslash S} x_\tau+d_{tl}$. It remains to be proved that $\sum_{\tau\in N_{t-1}\backslash S} x_\tau$ equals $d_{1,t-1}$. Suppose $I_{t-1}>0$. From the minimality of t, it follows that there are no production periods in $S\cap N_{t-1}$, and therefore we can reduce the production costs by transferring production from the last production period in N_{t-1} to t.

iii) If $t\in S\backslash R$, then let $s\in\{p(t)+1,\ldots,t\}$ be the first period with $z_s=1$. By the

minimality of t we have the following.

a) For $\tau<t,\ \tau\in R,\ y_\tau=0.$

b) For $\tau<t,\ \tau\in S\backslash R,\ z_{p(\tau)+1},\ldots,z_\tau=0.$

Moreover, since t is minimal $\{s,\ldots,t-1\}\cap S=\emptyset$, and therefore $p(t)<s$. Since $\beta_s=\ldots=\beta_t=\alpha_t=0$, the production of d_{tl} can take place at no additional cost in t. Therefore

c) For $\tau>t,\ \tau\in N_l\backslash S,\ x_\tau=0,$ since $\alpha_\tau>0.$

d) For $\tau>t,\ \tau\in R,\ y_\tau=0,$ since $\beta_\tau>0.$

e) For $\tau>t,\ \tau\in S\backslash R,\ z_{p(\tau)+1}=\ldots=z_\tau=0,$ since $\gamma_{p(\tau)+1},\ldots,\gamma_\tau>0.$

Note that $z_{s+1},\ldots,z_t=0$, for otherwise the costs can be reduced by setting $y_{s+1}=\ldots=y_t=1$. By the minimality of s, it follows that $z_{p(t)+1}=\ldots=z_{s-1}=0.$

Since the coefficient of z_s is d_{tl}, the value of the left-hand side of the (l,R,S)-inequality is reduced to the quantity $\sum_{\tau\in N_{t-1}\backslash S} x_\tau+d_{tl}$. It remains to be proved that $\sum_{\tau\in N_{t-1}\backslash S} x_\tau$ equals $d_{1,t-1}$. Suppose $I_{t-1}>0$. From the minimality of t, it follows that there are no production periods in $S\cap N_{t-1}$, and therefore we can reduce the production costs by transferring production from the last production period in N_{t-1} to t.

This ends the proof of theorem 6.3.1. The remainder of this section is devoted to proving that the major part of the (l,R,S)-inequalities are facet-defining inequalities.

6.3.2. Facet-defining (l,R,S)-inequalities

Many of the (l,R,S)-inequalities are facet-defining. Before we show this the dimension of the convex hull of ELSS is to be determined.

The number of variables is $3T$. By the assumption that demands are positive, the first period is a production period, and therefore $z_1=1$ and $y_1=1$. Moreover, overproduction is not allowed, i.e., $\sum_{t=1}^{T} x_t = d_{1T}$. Thus, there are

three obvious equations, that must be satisfied by all feasible solutions. Therefore, the dimension of the convex hull of ELSS is at most $3(T-1)$.

It is left to be shown that the dimension of the convex hull of ELSS is at least $3(T-1)$. This is proved by exhibiting this number of linearly independent directions, each direction being the difference of two feasible solutions. For each period t $(t=2,\ldots,T)$, three directions will be exhibited. These three directions are: $z_t=1$; $y_t=1$ (all other variables are zero), and $x_t=1$, $x_1=-1$ (all other variables are zero). For each direction the two feasible solutions are given in order of appearance.

Direction	*First solution*	*Second solution*
$z_t=1$:	$z_1=1, y_1=1, x_1=d_{1T}$; $z_t=1$.	$z_1=1, y_1=1, x_1=d_{1T}$.
$y_t=1$:	$z_1=1, y_1=1, x_1=d_{1T}$; $z_t=1, y_t=1$.	$z_1=1, y_1=1, x_1=d_{1T}$; $z_t=1$.
$x_t=1; x_1=-1$:	$z_1=1, y_1=1, x_1=d_{1T}-1$; $z_t=1, y_t=1, x_t=1$.	$z_1=1, y_1=1, x_1=d_{1T}$; $z_t=1, y_t=1$.

It follows that verifying whether a valid inequality is facet-defining can be done by exhibiting $3(T-1)-1$ directions. This fact is used in the following theorem, where the facet-defining (l,R,S)-inequalities are identified.

6.3.2. Theorem.

Suppose that the demands d_t $(1\le t\le T)$ are positive. The (l,R,S)-inequalities define facets, if and only if the following conditions hold.

1. $1\notin S\neq\emptyset$.
2. $l<T$ *or* $|R|=1$.

Proof. First, suppose that condition 1 does not hold. If $1\in S$, then y_1 or z_1 appears in the left-hand side of the (l,R,S)-inequality, with coefficient d_{1l}. Thus, since $y_1=z_1=1$, by definition, all other variables in the left-hand side must be zero for each feasible solution satisfying the inequality as

equality. This implies that there can not be any other variables in the left-hand side. If $S=\emptyset$, then the inequality is $\sum_{t=1}^{l} x_t \geq d_{1l}$. Thus, $l=T$ gives the model equation 6.3.1. If $l<T$, then each solution satisfying this inequality as equality, also satisfies $y_{l+1}=1$ as equality, due to the condition that $d_{l+1}>0$. Thus, then it is not a facet either. In the following condition 1 is satisfied. For the sake of completeness, it should be mentioned that in general $\sum_{t=1}^{l} x_t \geq d_{1l}$ is a facet, when $d_{l+1}=0$.

First, suppose that $l<T$. We exhibit $3(T-2)+2$ linear independent directions each satisfying the (l,R,S)-inequality as equality. For each period $t\in\{2,\ldots,T\}\backslash\{l+1\}$, three directions are exhibited, and for period $l+1$, two directions are exhibited. Let $t\in\{l+2,\ldots,T\}$.

Direction	*First solution*	*Second solution*
$z_t=1$:	$z_1=1, y_1=1, x_1=d_{1l}$;	$z_1=1, y_1=1, x_1=d_{1l}$;
	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$;	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.
	$z_t=1$.	
$y_t=1$:	$z_1=1, y_1=1, x_1=d_{1l}$;	$z_1=1, y_1=1, x_1=d_{1l}$;
	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$;	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$;
	$z_t=1, y_t=1$.	$z_t=1$.
$x_t=1; x_{l+1}=-1$:	$z_1=1, y_1=1, x_1=d_{1l}$;	$z_1=1, y_1=1, x_1=d_{1l}$;
	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}-1$;	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$;
	$z_t=1, y_t=1, x_t=1$.	$z_t=1, y_t=1$.

Let $t=l+1$. Choose the first element $\tau\in S$, thus $\tau\in R$. Note that this is the only place where the assumption $S\neq\emptyset$ is used.

Direction	*First solution*	*Second solution*
$z_{l+1}=1$:	$z_1=1, y_1=1, x_1=d_{1,\tau-1}$;	$z_1=1, y_1=1, x_1=d_{1l}$;
	$z_\tau=1, y_\tau=1, x_\tau=d_{\tau,T}$;	$z_\tau=1, y_\tau=1, x_\tau=d_{\tau,T}$;
	$z_{l+1}=1$.	

$y_{l+1}=1$:	$z_1=1, y_1=1, x_1=d_{1,\tau-1}$;	$z_1=1, y_1=1, x_1=d_{1l}$;
	$z_\tau=1, y_\tau=1, x_\tau=d_{\tau,T}$;	$z_\tau=1, y_\tau=1, x_\tau=d_{\tau,T}$;
	$z_{l+1}=1, y_{l+1}=1$;	$z_{l+1}=1$.

For $t\in\{2,\ldots,l\}$ we distinguish four cases. First, let $\tau\in S$ be such that $t\in\{p(\tau)+1,\ldots,\tau\}$, if such a period τ exists.

CASE 1: $t\notin S$; $\tau\in R$, or τ does not exist.

Direction	*First solution*	*Second solution*
$z_t=1$:	$z_1=1, y_1=1, x_1=d_{1l}$;	$z_1=1, y_1=1, x_1=d_{1l}$;
	$z_t=1$;	
	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.
$y_t=1$:	$z_1=1, y_1=1, x_1=d_{1l}$;	$z_1=1, y_1=1, x_1=d_{1l}$;
	$z_t=1, y_t=1$;	$z_t=1$;
	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.
$x_t=1, x_1=-1$:	$z_1=1, y_1=1, x_1=d_{1l}-1$;	$z_1=1, y_1=1, x_1=d_{1l}$;
	$z_t=1; y_t=1, x_t=1$;	$z_t=1; y_t=1$;
	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.

CASE 2: $t\notin S$; $\tau\in S\backslash R$.

Direction	*First solution*	*Second solution*
$z_t=1$;	$z_1=1, y_1=1, x_1=d_{1,\tau-1}$;	$z_1=1, y_1=1, x_1=d_{1l}$;
$y_t,\ldots,y_\tau=1$;	$z_t, y_t,\ldots,y_\tau=1, x_\tau=d_{\tau l}$;	
$x_\tau=d_{\tau l}, x_1=-d_{\tau l}$	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.
$y_t=1$:	$z_1=1, y_1=1, x_1=d_{1,p(\tau)-1}$;	$z_1=1, y_1=1, x_1=d_{1,p(\tau)-1}$;
	$z_{p(\tau)}=1$;	$z_{p(\tau)}=1$;
	$y_{p(\tau)},\ldots,y_t=1$;	$y_{p(\tau)},\ldots,y_{t-1}=1$;
	$x_{p(\tau)}=d_{p(\tau),l}$;	$x_{p(\tau)}=d_{p(\tau),l}$;
	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.

$x_t=1, x_1=-1$:	$z_1=1, y_1=1, x_1=d_{1,\tau-1}-1$;	$z_1=1, y_1=1, x_1=d_{1,\tau-1}$;
	$z_t, y_t, \ldots, y_\tau=1, x_\tau=d_{\tau l}, x_t=1$	$z_t, y_t, \ldots, y_\tau=1, x_\tau=d_{\tau l}$;
	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.

CASE 3: $t\in S\backslash R$.

$z_t=1$;	$z_1=1, y_1=1, x_1=d_{1,t-1}$;	$z_1=1, y_1=1, x_1=d_{1l}$;
$y_t=1$;	$z_t=1; y_t=1, x_t=d_{tl}$;	
$x_1=-d_{tl}, x_t=d_{tl}$:	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$;	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.
$y_t=1$:	$z_1=1, y_1=1, x_1=d_{1,p(\tau)-1}$;	$z_1=1, y_1=1, x_1=d_{1,p(\tau)-1}$;
	$z_{p(\tau)}=1$;	$z_{p(\tau)}=1$;
	$y_{p(\tau)}, \ldots, y_t=1$;	$y_{p(\tau)}, \ldots, y_{t-1}=1$;
	$x_{p(\tau)}=d_{p(\tau),l}$;	$x_{p(\tau)}=d_{p(\tau),l}$;
	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.
$x_t=1, x_{p(t)}=-1$:	$z_1=1, y_1=1, x_1=d_{1,p(\tau)-1}$;	$z_1=1, y_1=1, x_1=d_{1,p(\tau)-1}$;
	$z_{p(\tau)}=1$;	$z_{p(\tau)}=1$;
	$y_{p(\tau)}, \ldots, y_t=1$;	$y_{p(\tau)}, \ldots, y_t=1$;
	$x_{p(\tau)}=d_{p(\tau),l}-1, x_t=1$;	$x_{p(\tau)}=d_{p(\tau),l}$;
	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.

CASE 4: $t\in R$.

Direction	*First solution*	*Second solution*
$z_t=1$:	$z_1=1, y_1=1, x_1=d_{1l}$;	$z_1=1, y_1=1, x_1=d_{1l}$;
	$z_t=1$;	
	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.
$y_t=1$,	$z_1=1, y_1=1, x_1=d_{1,t-1}$;	$z_1=1, y_1=1, x_1=d_{1l}$;
$x_t=d_{tl}, x_1=-d_{tl}$:	$z_t=1; y_t=1, x_t=d_{tl}$;	$z_t=1$;
	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.
$x_t=1, x_{l+1}=-1$:	$z_1=1, y_1=1, x_1=d_{1,t-1}$;	$z_1=1, y_1=1, x_1=d_{1,t-1}$;
	$z_t=1; y_t=1, x_t=d_{tl}+1$;	$z_t=1; y_t=1, x_t=d_{tl}$;
	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}-1$.	$z_{l+1}=1, y_{l+1}=1, x_{l+1}=d_{l+1,T}$.

All the exhibited directions are linearly independent. For each $t\in\{2,\ldots,T\}$, there is exactly one direction for which z_t is nonzero. Considering the directions that are left over, for each $t\in\{2,\ldots,T\}$, there is exactly one direction for which y_t is nonzero. For each $t\in\{1,\ldots,l,l+2,\ldots,T\}\backslash S$, there is exactly one direction in which x_t is nonzero. For all $t\in S\backslash R$, there are directions with x_t and $x_{p(t)}$ nonzero. Since $p(t)<t$, these directions are also linear independent from the others, as can be seen by considering the periods in $S\backslash R$ in decreasing order. Finally, for each $t\in R$, there is a direction with x_t nonzero, and thus all exhibited directions are linear independent.

This leaves the case $l=T$.

If $|R|=1$, then the directions in the cases 1 to 4 can be used, where the parts in which the periods $l+1$ and higher are omitted. This holds with one exception, i.e., the last direction in case 4, which should be omitted completely. However, since $|R|=1$, this is only one direction. Therefore, there are $3(T-2)+2$ directions left over.

If $|R|>1$, then the $(T,R,S)-$inequality is not facet-defining. This is proved as follows. Let u be the last period in R. We define $R'=\{u\}$ and $S'=S\cap\{u,\ldots,T\}$. We will prove that each feasible solution that satisfies the $(T,R,S)-$inequality as equality, also satisfies the $(T,R',S')-$inequality as equality. So, take a feasible solution satisfying the $(T,R,S)-$inequality as equality. First, suppose that for each $t\in R$ we have $y_t=0$, and for each $t\in S\backslash R$ we have $z_{p(t)+1}+\ldots+z_t=0$. Then there is no production in S, and therefore the variables x_t $(t\in S)$ can be added to the left-hand side of the $(T,R,S)-$inequality without violating equality. This new inequality is clearly dominated by the $(T,R',S')-$inequality, since each variable has a smaller or equal coefficient in the left-hand side of the latter. Second, suppose that for some $t\in R$ we have $y_t=1$, or for some $t\in S\backslash R$ we have $z_{p(t)+1}+\ldots+z_t\geq 1$. Choose t minimal with this property. If $t\in S'$, then no production takes place in $S\backslash S'$, by the minimality of t. Addition of the variables x_τ $(\tau\in S\backslash S')$ leads to an inequality dominated by the $(T,R',S')-$inequality. This leaves the case $t\notin S'$. Then $t<u$. Note that the coefficient of the variable y_t (if $t\in R$) or of $z_{p(t)+1}+\ldots+z_t$ (if $t\in S\backslash R$) is d_{tT}. Moreover, by the minimality of t, the

production in the periods $\{1,\ldots,t-1\}\backslash S$ is at least $d_{1,t-1}$. Since this already contributes at least d_{1T} we must have $y_u = 0$, and $z_{p(\tau)+1} + \ldots + z_\tau = 0$ for $\tau \in S'\backslash R'$. Finally, since no production takes place in S', this implies the following: $\sum_{\tau \in N_T \backslash S'} x_\tau = \sum_{\tau \in N_T^\tau} x_\tau = d_{1T}$.

□

6.3.3. Separation for the (l,R,S)-inequalities

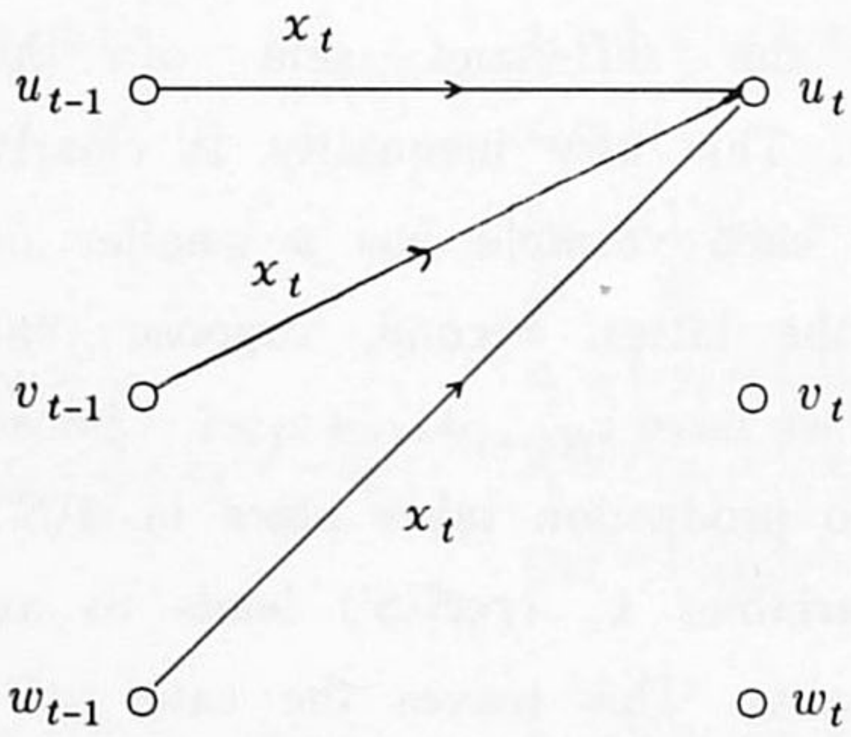
Here, we show that the separation algorithm for the (l,R,S)-inequalities can be formulated as a shortest path problem.

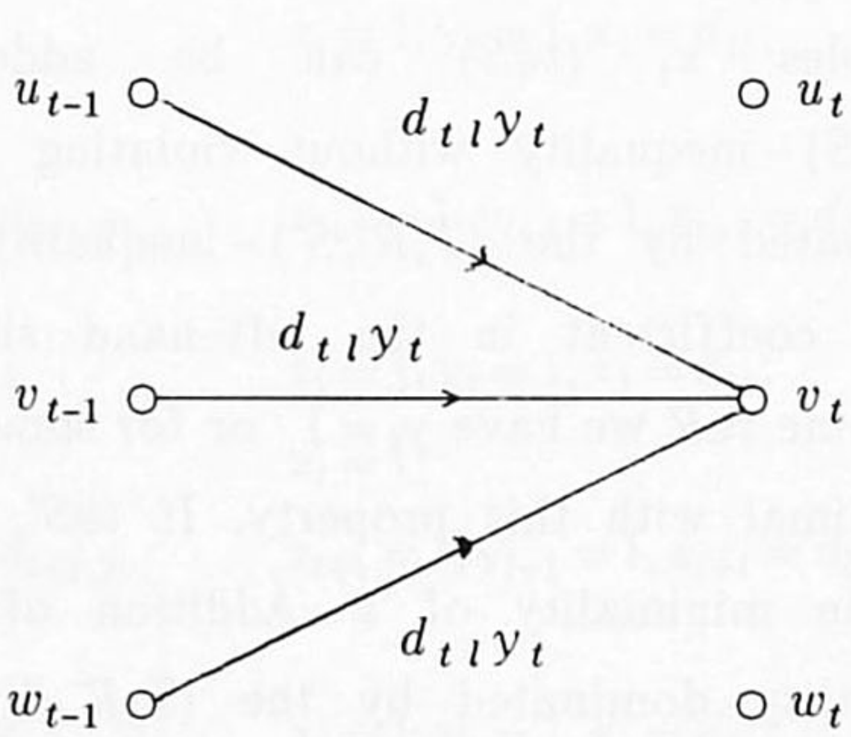
We fix l. Then we define three nodes for each period $t \in \{0,\ldots,l\}$: u_t, v_t and w_t. Moreover, a starting node n_0 and an ending node n_l are defined. There are three arcs with n_0 as a tail: (n_0,u_0), (n_0,v_0) and (n_0,w_0) all with zero costs. Moreover, there are three arcs with head n_l: (u_l,n_l), (v_l,n_l) and (w_l,n_l), also with zero costs. To model the (l,R,S)-inequalities in a network we define three types of arcs.

Type 1: arcs (u_{t-1},u_t), (v_{t-1},u_t), (w_{t-1},u_t) with cost x_t.

Type 2: arcs (u_{t-1},v_t), (v_{t-1},v_t), (w_{t-1},v_t) with cost $d_{tl}y_t$.

Type 3: arcs $(v_{p(t)},w_t)$, $(w_{p(t)},w)$ with cost $\sum_{\tau=p(t)+1}^{t-1} x_\tau + d_{tl} \sum_{\tau=p(t)+1}^{t} z_\tau$.





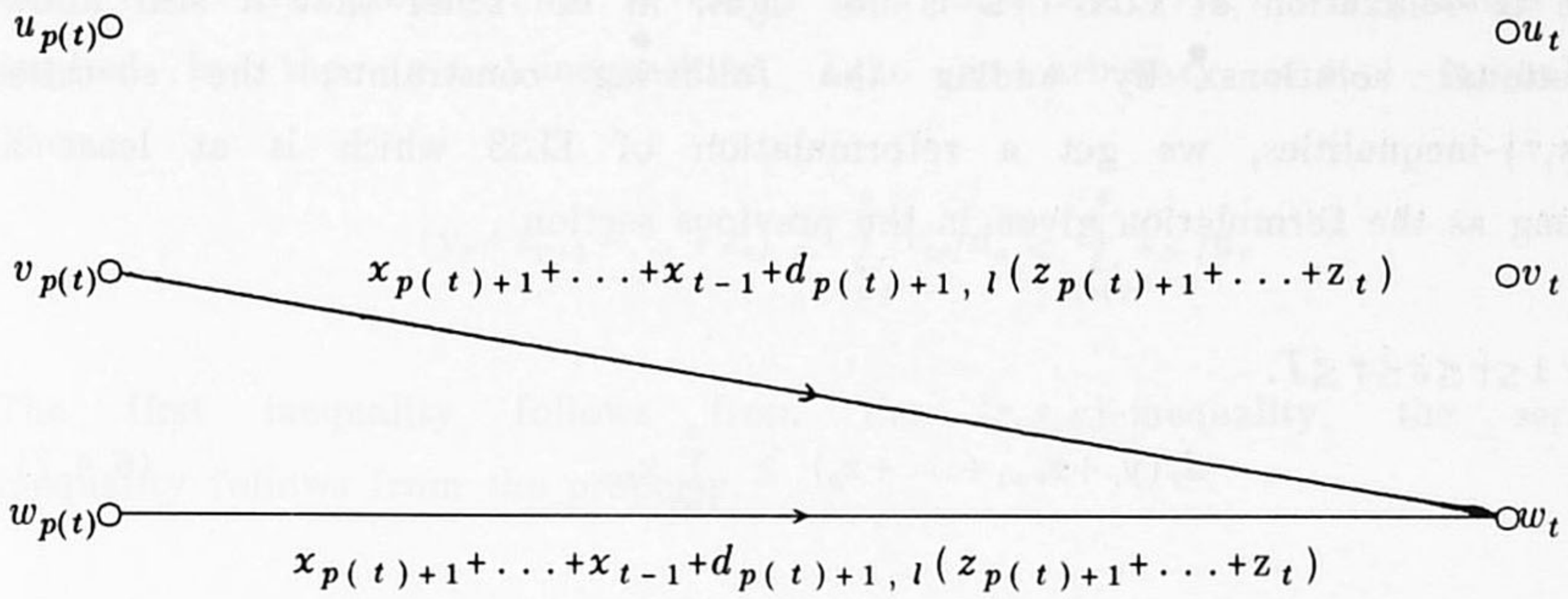


Figure 6.3. *Shortest path network for separation.*

Each path in the network corresponds to the left-hand side of a unique (l,R,S)-inequality. In particular, the nodes $\{v_t\}$ and $\{w_t\}$ correspond to the sets R and $S\backslash R$, respectively. Therefore, the shortest path in the network can be compared with d_{1l}, in order to find a violated (l,R,S)-inequality for a fixed l.

There are $O(l^2)$ arcs in the network, and since it is acyclic the shortest path problem in the network can be solved in $O(l^2)$ time. Doing so, for each $l \leq n$ gives an $O(T^3)$ algorithm, to find the most violated (l,R,S)-inequality. This is to be compared with the single max-flow calculation on a graph with $O(T^2)$ nodes derived in Rardin and Wolsey [41].

6.4. The Uncapacitated Facility Location Reformulation of ELSS

In this section, we consider the uncapacitated facility location reformulation of ELSS. It is modeled as follows.

$$\text{(ELSS-UFL)} \quad \min \sum_{t=1}^{T} \left(g_t z_t + f_t y_t + c_t\left(\sum_{\tau=t}^{T} x_{t\tau}\right)\right) \tag{6.4.1}$$

$$\text{s.t.} \quad \sum_{t=1}^{\tau} x_{t\tau} = d_\tau \qquad (1 \leq \tau \leq T) \tag{6.4.2}$$

$$x_{t\tau} \leq d_\tau y_t \qquad (1 \leq t \leq \tau \leq T) \tag{6.4.3}$$

$$y_t \leq y_{t-1} + z_t \quad (y_0 = 0) \qquad (1 \leq t \leq T) \tag{6.4.4}$$

$$y_t, z_t \in \{0,1\} \qquad (1 \leq t \leq T) \tag{6.4.5}$$

$$x_{t\tau} \geq 0 \qquad (1 \leq t \leq \tau \leq T) \tag{6.4.6}$$

The LP-relaxation of ELSS-UFL is not tight, in the sense that it still allows fractional solutions. By adding the following constraints, the so-called (r,s,τ)-inequalities, we get a reformulation of ELSS which is at least as strong as the formulation given in the previous section.

Let $1 \leq r \leq s \leq \tau \leq T$.

$$d_\tau(y_r + z_{r+1} + \ldots + z_s) \geq \sum_{t=r}^{s} x_{t\tau} \qquad (6.4.7)$$

These inequalities can be found in Wolsey [57]. Validity follows directly from the multicommodity flow formulation, see section 6.2, by taking the cut consisting of the vertices $u_{r+1},\ldots,u_s,v_r,\ldots,v_s$. The inflow in this set is bounded from above by $d_\tau(y_r+z_{r+1}+\ldots+z_s)$, and the outflow is $\sum_{t=r}^{s} x_{t\tau}$. The proof that these inequalities, together with the inequalities 6.4.1,...,6.4.4, are at least as strong as the (l,R,S)-inequalities, constitutes the main part of this section. The hardest part of the proof has been done in section 6.2. It remains to be proved that the inequalities 6.4.8 are implied by the (r,s,τ)-inequalities. For a given τ take $1 \leq r_1 \leq s_1 < r_2 \ldots < r_{K-1} \leq s_{K-1} < r_K \leq \tau$, and let M be the union of the batches $\{r_k,\ldots,s_k\}$ $(1 \leq k \leq K)$.

$$\sum_{t \in N_\tau \setminus M} x_{t\tau} + d_\tau \sum_{k=1}^{K} y_{r_k} + d_\tau \sum_{k=1}^{K} (z_{r_k+1} + \ldots + z_{s_k}) \geq d_\tau \qquad (6.4.8)$$

Summing all (r_k,s_k,τ)-inequalities for $k=1,\ldots,K$ gives the following inequality.

$$d_\tau \sum_{k=1}^{K} y_{r_k} + d_\tau \sum_{k=1}^{K} (z_{r_k+1} + \ldots + z_{s_k}) \geq \sum_{t \in M} x_{t\tau} \qquad (6.4.9)$$

Addition of $\sum_{t \in N_\tau \setminus M} x_{t\tau}$ to both sides of the inequality, and substitution of $\sum_{t \in N_\tau} x_{t\tau}$ by d_τ. gives the desired inequality.

The number of constraints in ELSS-UFL, together with the (r,s,τ)-inequalities is $O(T^3)$. This number can be reduced to $O(T^2)$, if the demands are positive. Then there exists an optimal solution to ELSS, in which for any t, the quantities $x_{t\tau}/d_\tau$ are non-increasing in τ. A simple exchange argument shows

this. With this property, it is easily seen that the (r,s,τ)-inequalities are implied by the (r,s,s)-inequalities. Take an arbitrary (r,s,τ)-inequality. Then

$$(y_r+z_{r+1}+\ldots+z_s) \geq \sum_{t=r}^{s} x_{ts}/d_s \geq \sum_{t=r}^{s} x_{t\tau}/d_\tau$$

The first inequality follows from the (r,s,s)-inequality, the second inequality follows from the property.

Chapter 7

POLYHEDRAL DESCRIPTION OF THE DISCRETE LOT SIZING AND SCHEDULING PROBLEM

7.1. Introduction

The subject of chapter 4 was the development of polynomial time dynamic programming techniques to solve the single-item Discrete Lot Sizing and Scheduling problem (DLS). The topic of this chapter is to look at (partial) linear descriptions of the single-item DLS. An early paper on this subject is Kuik et al. [28]. Similar work on the closely related discrete lot sizing and scheduling problem with set-up costs (DLSS) has been done by Magnanti and Vachani [32], and Sastry [46].

It is well-known that binary linear programming problems are much harder to characterize, than their integer equivalents. For instance, the facet-defining inequalities for the zero/one knapsack are much more complicated, than the facet-defining inequalities of the integer knapsack. Such a phenomenon is also encountered, when the polyhedral description of DLS is regarded. Compared with ELS and ELSS, where there is only one class of non-trivial inequalities the situation with respect to DLS is much more complicated. There is a wide variety of classes of facets with zero/one coefficients. However, it is not known whether these are all the facets with zero/one coefficients. Besides this, it is also not known whether there are more intricate classes of facets, with other than zero/one coefficients.

In section 7.2, the model for DLS and its LP-relaxation will be reviewed. Then the dimension of the convex hull of feasible solutions is determined, and the (partial) polyhedral description of DLS is given. We will describe some

classes of valid inequalities. For one class we will give necessary and sufficient conditions under which these inequalities are facets. For the other classes, there is a gap between the necessary and sufficient conditions as shown in subsection 7.2.4. Unfortunately, there is no separation algorithm available for any one of the described classes of valid inequalities. Besides the results of section 7.2, another partial result is described in Kuik et al. [28]. Here instances of DLS are identified, which always yield integer solutions under the constraints of the model. In section 7.3, we will give a facility location alike model for DLS. In this model the production variables and the start-up variables are disaggregated. This model is shown to be equivalent to a min cost flow formulation, and therefore it yields a complete polyhedral description.

7.2. Facets for DLS

The linear programming relaxation of DLS is the following.

$$\text{(RDLS)} \qquad \min \sum_{t=1}^{T} (g_t z_t + c_t x_t) \tag{7.2.1}$$

$$\text{s.t.} \qquad \sum_{\tau=1}^{T} x_\tau = d_{1T} \tag{7.2.2}$$

$$\sum_{\tau=1}^{t} x_\tau \geq d_{1t} \qquad (1 \leq t \leq T-1) \tag{7.2.3}$$

$$x_t \leq x_{t-1} + z_t \qquad (x_0 := 0) \qquad (1 \leq t \leq T) \tag{7.2.4}$$

$$0 \leq x_t \leq 1 \qquad (1 \leq t \leq T) \tag{7.2.5}$$

$$0 \leq z_t \leq 1 \qquad (1 \leq t \leq T) \tag{7.2.6}$$

Note that 7.2.3 enforces that the cumulative demand in the first t periods is also produced in these periods, i.e., backlogging is not allowed. Equation 7.2.2 prevents overproduction, and finally 7.2.4 ensures that each set of consecutive production periods (a so-called batch) begins with a start-up.

In the remainder of this section, different types of facets are described. Since the dimension of the convex hull is of crucial importance in the proofs,

an analysis of the dimension of the convex hull is performed in the following subsection.

7.2.1. Dimension of DLS

For a given instance of DLS, it is not straightforward to determine the dimension of the convex hull, since it depends on the demand function. However, it will be shown that a few simplifying assumptions suffice to get rid of the demand dependency. Moreover, it is fairly easy to get a full dimensional polyhedron. Suppose that we are given a T-period instance with demand function d, i.e., d_t $(1 \leq t \leq T)$ denotes the demand in period t. There are $2T$ variables and there is one clear equation: $\sum_{\tau=1}^{T} x_\tau = d_{1T}$. Thus the dimension is at most $2T-1$.

The actual dimension depends on the following properties of the demand function. If $d_1 = 1$, then $z_1 = x_1 = 1$ must hold. In general, if $d_1 = d_2 = \ldots = d_t = 1$, then $z_1 = x_1 = x_2 = \ldots = x_t = 1$, which yields another $t+1$ equations. In the remainder of this section, we eliminate this demand dependency by assuming that $d_1 = 0$. For any particular instance, this can be achieved by ignoring the periods before the first non-demand period. We can also add a new first period with no demand. A second demand dependency arises when $d_T = 0$, since this implies $x_T = 0$. In general, this applies for all periods after the last demand period t, if it precedes T. In this case, there are $T-t$ obvious equations, i.e., $x_{t+1} = \ldots = x_T = 0$. This demand dependency can be eliminated by supposing that $d_T = 1$. Under the two given assumptions the dimension of the convex hull of DLS is not dependent on the demands anymore, as follows from the following theorem.

7.2.1. Theorem.

The convex hull of a T-period instance of DLS has dimension $2T-1$, under the assumptions $d_1 = 0$ and $d_T = 1$.

Proof. A basic solution is created, where we distinguish between demand periods and non-demand periods. For each $t \in \{1, \ldots, T\}$, we define $z_t = 1$ and $x_t = d_t$. This basic solution is used extensively, to create $2T-1$ linear

independent directions. In the following we will only exhibit the values of variables that differ from the basic solution. First, T directions with $z_t = 1$ $(1 \leq t \leq T)$ are created. Second, $T-1$ directions with $x_t = 1$ $(1 < t \leq T)$ are created.

For a period t with $d_t = 0$, the first solution is the basic solution, and the second solution is the basic solution, where z_t is changed from value 1 to value 0. This creates the direction $z_t = 1$.

For a period t with $d_t = 1$, the first solution is the basic solution, and the second solution is the basic solution, where z_t is changed from value 1 to value 0, x_t is changed from value 1 to value 0, and x_1 is changed from value 0 to value 1. This creates the direction $z_t = 1$, $x_t = 1$, $x_1 = -1$.

The first T directions are linear independent of the others, since they are the only ones that contain z_t $(1 \leq t \leq T)$ as a non-zero component.

For a period t with $d_t = 0$ $(t \neq 1)$, the first solution is the basic solution, where x_t is changed from value 0 to 1 and x_T is changed from value 1 to 0. The second solution is the basic solution. This creates the direction $x_t = 1$, $x_T = -1$.

For a period t with $d_t = 1$ $(t \neq T)$, the first solution is the basic solution, and the second solution is the basic solution, where x_t is changed from value 1 to value 0, and x_1 is changed from value 0 to value 1. This creates the direction $x_t = 1$, $x_1 = -1$.

These are $T-2$ directions that contain $x_t = 1$ $(1 < t < T)$, and either $x_1 = -1$ $(d_t = 1)$ or $x_T = -1$ $(d_t = 0)$.

Finally, the direction $x_T = 1$, $x_1 = -1$ is created by taking the basic solution as the first solution, and by taking the basic solution, where x_T is changed from 1 to 0, and x_1 is changed from 0 to 1.

□

A final assumption made in the following, is the relaxation of the constraint 7.2.2 to $\sum_{\tau=1}^{T} x_\tau \geq d_{1T}$, i.e., overproduction is allowed. This relaxation ensures

that the polyhedron becomes full dimensional. This is proved in theorem 7.2.2. The major advantage of this relaxation is that the facet proofs become more transparent, since the feasible solutions forming the directions can be created more easily. This is due to the fact that the solution in which all variables have value one is feasible under the relaxation of constraint 7.2.2. This solution will be used in the following theorem, also. Another advantage is that the facet-defining inequalities become unique.

7.2.2. Theorem.

The convex hull of a T-period instance of DLS has dimension $2T$, under the assumptions that $d_1 = 0$ and overproduction is allowed.

Proof. The proof falls into two cases. One case where the direction $z_t = 1$ (all variables not mentioned have value zero) is established, and one where the direction $x_t = 1$ is established. Both cases use feasible solutions which have a large part in common. This part consists of the values of the variables of periods other than period t. For each $\tau \neq t$, we fix $z_\tau = x_\tau = 1$.

Case 1: direction $z_t = 1$ $(1 \leq t \leq T)$.

In the first solution we take $z_t = 1$; $x_t = 0$, and in the second solution we take $z_t = 0$; $x_t = 0$.

Note that this results in feasible solutions. Since $d_1 = 0$, and period t is the only period without production, all demands can be fullfilled, i.e., the inequalities 7.2.3 are satisfied.

Case 2: direction $x_t = 1$ $(1 \leq t \leq T)$.

In the first solution we take $z_t = 1$; $x_t = 1$, and in the second solution we take $z_t = 1$; $x_t = 0$.

Since we have exhibited $2T$ linear independent directions, this ends the proof. □

A corollary of the two theorems is that the original polytope is a facet of the full polyhedron, which follows trivially by comparing the dimensions of both polytopes.

This subsection ends with two examples of the implications of replacing the equation 7.2.2, by an inequality. The first example shows that facets of the full-dimensional polyhedron need not be facets of the original polyhedron. The second example shows that facets of the original polyhedron are not uniquely representable, i.e., there are representations which are not valid for the full polyhedron.

EXAMPLE 1: $T=4$: $d_1=d_2=d_3=0; d_4=1$.

The inequality $x_1+z_2+x_3+z_4\geq 1$ is a facet of the full polyhedron, but it is a lower dimensional face of the original one. The latter follows, since each feasible solution satisfying $x_1+z_2+x_3+z_4=1$ also satisfies $z_4=x_4$. Note that $z_4\geq x_4$ is valid, if 7.2.2 holds as an equation, but not if it is relaxed. In general, the inequality $z_t\geq x_t$ is valid under the condition $d_{t-1,T-1}=0$, in the original polyhedron but not in the full polyhedron.

EXAMPLE 2: $T=6$: $d_1=d_2=d_3=d_4=0; d_5=d_6=1$.

The inequality $z_3+z_4\geq x_4$ is a facet in the original polyhedron, but it is not valid in the full polyhedron, because overproduction is allowed there. To make it a facet of the full polyhedron, the equation $x_1+x_2+x_3+x_4+x_5+x_6=2$ must be added. Then we obtain the inequality $x_1+x_2+x_3+z_3+z_4+x_5+x_6\geq 2$. In general, if $d_{t_1}=0$ and $d_{t_1+1,T}=t_2-t_1$, then $z_{t_1+1}+\ldots+z_{t_2}\geq x_{t_2}$ is a facet in the original polyhedron, but it is not valid in the full polyhedron. The equation $\sum_{t-1}^{T} x_t=d_{1T}$ should be added to obtain the following facet for the full polyhedron.

$$x_1+\ldots+x_{t_2-1} + z_{t_1+1}+\ldots+z_{t_2} + x_{t_2+1}+\ldots+x_T \geq d_{1T}$$

7.2.2. Facet-defining inequalities in the model

In this subsection the facet-defining inequalities that are part of the model are evaluated, under the assumptions made in the preceding subsection. These assumptions are

A1) $d_1=0$ A2) $d_T=1$ A3) $\sum_{\tau=1}^{T} x_\tau \geq d_{1T}$

Moreover, the general type of inequalities with 0/1-coefficients is defined, and conditions that restrict the search for facets are defined.

Variable bounding inequalities

These are the inequalities that bound the variables in the model:

$$x_t \geq 0 \ (1 \leq t \leq T); \qquad z_t \geq 0 \ (1 \leq t \leq T);$$
$$x_t \leq 1 \ (1 \leq t \leq T); \qquad z_t \leq 1 \ (1 \leq t \leq T).$$

7.2.3. Theorem.

a) The inequalities $x_t \geq 0 \ (1 \leq t \leq T)$ *define facets, except if* $t=1$ *and* $d_2=1$, *or if* $t>1$ *and* $d_2=\ldots=d_t=1$.

b) The inequalities $x_t \leq 1 \ (1 \leq t \leq T)$ *define facets, except if* $t=1$.

c) The inequalities $z_t \geq 0 \ (1 \leq t \leq T)$ *define facets, except if* $t=1$, *or if* $t>1$ *and* $d_2=\ldots=d_t=1$.

d) The inequalities $z_t \leq 1 \ (1 \leq t \leq T)$ *define facets for all* t.

Proof. The basic feasible solution used to determine the directions has all variables equal to 1, i.e., $z_\tau=1$; $x_\tau=1 \ (1 \leq \tau \leq T)$. In the following, only the differences with this basic feasible solution are mentioned. Thus, these are the variables with value zero.

a) $x_t \geq 0$. If $d_2=1$, then $x_1=0$ implies $x_2=1$. If $d_2=\ldots d_t=1$, then $x_t=0$ implies $\sum_{\tau=1}^{t} x_\tau = \sum_{\tau=1}^{t-1} x_\tau = d_{1t}$. Thus, in these cases $x_t \geq 0$ is not a facet. For the other cases the direction will be exhibited in the following.

Direction	*First solution*	*Second solution*
$z_\tau=1, x_\tau=1 \ (\tau=1,t+1)$:	$x_t=0$;	$x_t=0, z_\tau=x_\tau=0$.
$z_\tau=1 \ (\tau \neq 1,t+1)$:	$x_t=0$;	$x_t=0, z_\tau=0$.
$x_\tau=1 \ (\tau \neq t)$:	$x_t=0$;	$x_t=0, x_\tau=0$.

b) $x_t \leq 1$. If $x_1=1$, then $z_1=1$, thus $t \neq 1$.

Direction	*First solution*	*Second solution*
$z_1=1$:	$x_1=0$;	$x_1=z_1=0$.

$z_\tau = 1$ $(\tau \neq 1)$:	$z_\tau = 0.$
$x_\tau = 1$ $(\tau \neq t)$:	$x_\tau = 0.$

c) $z_t \geq 0$. If $z_1 = 0$, then $x_1 = 0$. If $d_2 = \ldots = d_t = 1$, then $z_t = 0$ implies $x_{t-1} = 1$. Note that otherwise $x_{t-1} = x_t = 0$, and thus $\sum_{\tau=1}^{t} x_\tau < d_{1t}$, a contradiction. In all other cases we can define the following directions.

Direction	*First solution*	*Second solution*
$z_1 = 1$:	$x_1 = z_t = x_t = 0;$	$x_1 = z_t = x_t = z_1 = 0.$
$z_\tau = 1$ $(\tau \neq 1, t)$:	$z_t = 0;$	$z_t = z_\tau = 0.$
$x_\tau = 1$ $(\tau \neq t-1)$:	$z_t = 0;$	$z_t = x_\tau = 0.$
$x_{t-1} = 1$:	$z_t = x_t = 0;$	$z_t = x_t = x_{t-1} = 0.$

d) $z_t \leq 1$.

Direction	*First solution*	*Second solution*
$z_1 = 1$:	$x_1 = 0;$	$z_1 = x_1 = 0.$
$z_\tau = 1$ $(\tau \neq 1, t)$:		$z_\tau = 0.$
$x_\tau = 1$:		$x_\tau = 0.$

All solutions mentioned in the cases a-d are feasible. In cases a and c, it should be noted that there are at least two periods in $\{1, \ldots, t\}$ without demand. Clearly, $2T-1$ linear independent directions have been created in each case.

□

Inequalities that force start-ups

These are the inequalities $z_1 \geq x_1$ and $x_{t-1} + z_t \geq x_t$ $(1 \leq t \leq T)$.

7.2.4. Theorem.

The inequality $z_1 \geq x_1$ is facet defining.

Proof. The directions created in the proof of theorem 7.2.2, can be used, except for the direction in which a feasible solution with $x_1 = 0$, $z_1 = 1$ is used.

□

7.2.5. Theorem.

The inequalities $x_{t-1}+z_t \geq x_t$ $(t \geq 2)$ *define facets, unless* $d_2 = \ldots = d_t = 1$.

Proof. Let t be such $d_2 = \ldots = d_t = 1$, and consider solutions for which $x_{t-1}+z_t = x_t$. Then $x_t = 1$. Otherwise, $x_{t-1} = x_t = 0$ which implies $\sum_{\tau=1}^{t} x_\tau < d_{1t}$. In the following we take the basic solution consisting of $z_t = 0$, $x_t = 1$, $z_\tau = x_\tau = 1$ $(\tau \neq t)$.

Direction	*First solution*	*Second solution*
$z_\tau = 1$ $(\tau \neq t-1, t)$:	$x_\tau = 0$;	$z_\tau = x_\tau = 0$.
$z_{t-1} = 1$:	$x_{t-1} = x_t = 0$;	$z_{t-1} = x_{t-1} = x_t = 0$.
$z_t = 1, x_{t-1} = -1$:	$z_t = 1, x_{t-1} = 0$.	
$x_\tau = 1$ $(\tau \neq t-1, t)$:		$x_\tau = 0$.
$x_{t-1} = x_t = 1$:		$x_{t-1} = x_t = 0$.

Clearly, $2T-1$ directions have been created. It is not difficult to see that these directions are linearly independent.

□

Production inequalities

These are the inequalities that ensure that there is enough production, i.e., the inequalities $\sum_{\tau=1}^{t} x_\tau \geq d_{1t}$ $(1 \leq t \leq T)$. The following theorem provides conditions under which these are facet-defining inequalities.

7.2.6. Theorem.

The inequalities $\sum_{\tau=1}^{t} x_\tau \geq d_{1t}$ *define facets, if* $d_t = 1$, *and* $d_{t+1} = 0$ *or* $t = T$.

Proof. If $d_t = 0$, then $\sum_{\tau=1}^{t} x_\tau = d_{1t}$ implies $\sum_{\tau=1}^{t-1} x_\tau = d_{1,t-1}$. If $d_{t+1} = 1$, then $\sum_{\tau=1}^{t} x_\tau = d_{1t}$ implies $x_{t+1} = 1$. Thus, the conditions are certainly necessary. Sufficiency can be concluded from the following. As a basic solution, we take $z_\tau = 1$ $(1 \leq \tau \leq T)$, and $x_\tau = d_\tau$ $(1 \leq \tau \leq t)$, $x_\tau = 1$ $(t+1 \leq \tau \leq T)$. For each period τ after t, the directions $z_\tau = 1$ and $x_\tau = 1$ are easily created. This leaves the directions for the periods $\{1, \ldots, t\}$.

	Direction	*First solution*	*Second solution*
$d_\tau = 0$ $(\tau \neq 1)$:	$z_\tau = 1$:		$z_\tau = 0$.

	$x_\tau = 1, x_t = -1$:	$x_\tau = 1, x_t = 0$.	
$d_\tau = 1$:	$z_\tau = 1$:	$x_\tau = 0, x_1 = 1$;	$z_\tau = x_\tau = 0, x_1 = 1$.
	$x_\tau = 1, x_1 = -1$:		$x_\tau = 0, x_1 = 1$.

It is not too hard to see that these directions are linear independent. In fact, the proof is analogous to the proof of theorem 7.2.1. □

In the subsections 7.2.3 and 7.2.4 we will consider inequalities with coefficients 0 and 1 for the production and start-up variables, i.e., these inequalities have the following structure.

$$\sum_{t \in X} x_t + \sum_{t \in Z} z_t \geq d_{XZ} \qquad (7.2.7)$$

where d_{XZ} is a positive integer and X and Z are subsets of $\{1,\ldots,T\}$. Feasible solutions that satisfy 7.2.7 as equality will be called optimal solutions in the following.

The only inequalities that are known not to satisfy the type of 7.2.7 are the variable upper bound inequalities $z_t \leq 1$ $(1 \leq t \leq T)$ and $x_t \leq 1$ $(1 \leq t \leq T)$, and the start-up inequalities $x_{t-1} + z_t \geq x_t$ $(1 \leq t \leq T)$. It is an open question whether inequalities not satisfying 7.2.7, other than the mentioned ones, exist. In the sequel we restrict ourselves to type 7.2.7 inequalities. We start with some properties of these inequalities that simplify the exposition.

P1) $1 \notin Z$. If $1 \in Z$, then $z_1 = x_1$ for all feasible solutions satisfying 7.2.7 as equality. Namely, a solution satisfying $z_1 = 1$ and $x_1 = 0$ can be improved by setting $z_1 = 0$.

P2) $Z \neq \emptyset$. If $Z = \emptyset$, then the inequality defines a facet if and only if it is a production inequality (see subsection 7.2.2). This is fairly easy to see, and therefore the proof is left to the reader.

Finally, the restriction $T \in X \cup Z$ is imposed on the valid inequalities of type 7.2.7. Although there are many facet-defining inequalities that do not satisfy

this restriction, these can be found using the methods in this subsection by regarding a restricted planning horizon, i.e., if t is the last period in $X \cup Z$ for some valid inequality of type 7.2.7, then it can be determined whether this inequality is facet-defining by ignoring the periods $\{t+1,\ldots,T\}$, under the same conditions used in the following for the full planning horizon.

7.2.7. Lemma.

Consider an inequality of type 7.2.7, *satisfying* P1 *and* P2, *such that* $t<T$ *is the last period in* $X \cup Z$.

i) *This inequality is valid with respect to* $\{1,\ldots,T\}$ *if and only if it is valid with respect to* $\{1,\ldots,t\}$.

ii) *This inequality is facet-defining with respect to* $\{1,\ldots,T\}$ *if and only if it is facet-defining with respect to* $\{1,\ldots,t\}$.

Proof. If the inequality is valid with respect to $\{1,\ldots,t\}$, then it is certainly valid with respect to $\{1,\ldots,T\}$, since the set of restrictions is only extended. If it is not valid with respect to $\{1,\ldots,t\}$, then take a feasible solution which violates the inequality. Extending this solution with $x_\tau = z_\tau = 1$ for $\tau > t$, creates a feasible solution with respect to $\{1,\ldots,T\}$ that also violates the inequality. This ends the proof of the first part of the lemma.

Suppose that the inequality is facet-defining with respect to $\{1,\ldots,t\}$. Then there are $2t-1$ linear independent directions using the variables x_τ, z_τ $(\tau \leq t)$ only. Moreover, there is a feasible solution satisfying the inequality as equality with $\sum_{\tau=1}^{t} x_\tau > d_{1t}$. Otherwise, all optimal feasible solutions would satisfy $\sum_{\tau=1}^{t} x_\tau = d_{1t}$, implying that the inequality is not a facet, since $Z \neq \emptyset$. Take this solution and extend it with $x_\tau = z_\tau = 1$ $(t < \tau \leq T)$. Using this solution as a basis the directions $x_\tau = 1$ $(t < \tau \leq T)$ and $z_\tau = 1$ $(t < \tau \leq T)$ can be created. For the direction $x_\tau = 1$, the basis is taken as a first solution, and the second solution is like the first but with $x_\tau = 0$. The latter solution is taken as a first solution to create the direction $z_\tau = 1$. The second solution has also $z_\tau = 0$. Thus, $2(T-t)$ new linear independent directions have been created, and therefore the inequality is also facet-defining with respect to $\{1,\ldots,T\}$.

Finally, suppose that the inequality is not facet-defining with respect to $\{1,\ldots,t\}$. Then there is a set of valid inequalities that dominate the given one. However, these inequalities are also valid with respect to $\{1,\ldots,T\}$, which has been proved in the first part of this lemma. Thus, the original inequality is also dominated with respect to the planning horizon $\{1,\ldots,T\}$ by these inequalities which implies that it is not a facet with respect to this planning horizon.

□

The following lemma shows that the choices for restricted planning horizons can be limited to periods $\{1,\ldots,t)$, where t is a demand period.

7.2.8. Lemma.

Consider a valid inequality of type 7.2.7, *such that* $t<T$ *is the last period in* $X\cup Z$. *If this inequality is facet-defining, then* $d_t=1$.

Proof. Suppose that $d_t=0$. If $t\in X$, then any feasible solution with $x_t=1$ can be improved by setting $x_t=0$ and $x_\tau=z_\tau=1$ for $\tau=t+1,\ldots,T$. This does not affect the feasibility of the solution, since $\sum_{\tau=1}^{t}x_\tau\geq\sum_{\tau=1}^{t-1}x_\tau\geq d_{1,t-1}=d_{1,t}$. Therefore, all feasible solutions that satisfy 7.2.7 as equality satisfy $x_t=0$ also. If $t\in Z$, then the proof proceeds analogously.

□

From lemmas 7.2.7 and 7.2.8 it follows that facet defining inequalities satisfy the assumptions A1-A3 and the properties P1-P2 for some restricted planning horizon $\{1,\ldots,t\}$ $(t\leq T)$.

In subsection 7.2.3 we discuss inequalities, where $X\cap Z=\emptyset$. For these inequalities, the hole/bucket-inequalities, necessary and sufficient conditions are derived. In subsection 7.2.4 we will derive some necessary conditions for general inequalities of type 7.2.7 to define facets. We will also provide some classes of facet-defining inequalities. However, these classes are not exhaustive.

7.2.3. Hole/bucket-inequalities.

In the following we consider inequalities of the form 7.2.7 with the additional requirement that X and Z have no periods in common. The main goal is to derive necessary and sufficient conditions under which these inequalities define facets. The left-hand side of the facet-defining inequalities can be partitioned into so-called holes and buckets due to the following lemma

7.2.9. Lemma.

A facet-defining inequality of type 7.2.7 with $X \cap Z = \emptyset$ *satisfies the following condition. If* $t \in Z$, *then* $t-1 \in X$ *or* $t-1 \in Z$.

Proof. Suppose that there is a period t with $t \in Z$ and $t-1 \in N_T \setminus (X \cup Z)$. Each feasible solution (x,z) with $z_t=1$ can be improved by putting $z_t=0$ and $z_{t-1}=x_{t-1}=1$, which implies that each optimal solution satisfies $z_t=0$, and thus the inequality is not a facet-defining one.

□

By lemma 7.2.9 the planning horizon can be partitioned in batches of periods $B_k=\{t_k,\ldots,u_k\}$ $(1 \le k \le K)$, such that $t_k \in X$, and $B_k \setminus \{t_k\} \subseteq Z$ and a set of periods $H=\{h_l \mid 1 \le l \le L\}=\{1,\ldots,T\} \setminus (X \cup Z)$. Thus, the variables in the left-hand side of 7.2.7 of B_k are $x_{t_k}+z_{t_k+1}+\ldots+z_{u_k}$. Such batches are called buckets, since the use of one or more of the periods in a bucket B_k for production contributes at least one unit to the value of the left-hand side of 7.2.7. On the other hand, in a period of H, a so-called hole, production can take place without any contribution to the left-hand side of 7.2.7.

EXAMPLE

t	1	2	3	4	5	6	7	8	9	10	11	12
d_t	0	0	0	1	0	1	1	1	0	1	1	1

$$\begin{array}{llllllllllll} & z_2+ & & z_4+ & & z_6+ & & z_8+ & & z_{10}+ & & z_{12} \\ x_1+ & & x_3+ & & x_5+ & & x_7+ & & x_9+ & & x_{11}+ & \end{array} \geq 4$$

$$\begin{array}{llllllllllll} & z_2+z_3+ & & & & z_6+z_7+ & & z_9+ & & z_{11}+z_{12} & \\ x_1+ & & x_4+x_5+ & & & & x_8+ & & x_{10}+ & & & \end{array} \geq 3$$

$$\begin{array}{llllllllllll} & & z_4+ & & z_6+ & & z_8+ & & z_{10}+ & & z_{12} \\ x_3+ & & x_5+ & & x_7+ & & x_9+ & & x_{11}+ & & & \end{array} \geq 3$$

$$\begin{array}{llllllllllll} & & & z_6+z_7+ & & z_9+ & & z_{11}+z_{12} & & & & \\ x_4+x_5+ & & & & x_8+ & & x_{10}+ & & & & & \end{array} \geq 2$$

$$\begin{array}{llllllllllll} & & z_4+ & & & & z_8+ & & z_{10}+ & & z_{12} & \\ x_1+ & x_3+ & & x_5+ & & x_7+ & & x_9+ & & x_{11}+ & & \end{array} \geq 3$$

Clearly, the right-hand side is the minimum number of buckets that must be used for production. Given a set of buckets, covering the demands of the periods $\{1,\ldots,T\}$ by production in the buckets, i.e., the determination of the right-hand side d_{XZ} in order to make the bucket-inequality a valid one, is done by use of the following greedy algorithm. The size of a bucket B is its number of periods. It is denoted by $|B|$.

Greedy algorithm.

The periods with demand are considered in an increasing order one by one. The production period for the demand from period t is chosen in a greedy way in the following order of preference.

a) If there is a hole in $\{1,\ldots,t\}$, then it is chosen to produce d_t.
b) If there is a bucket in $\{1,\ldots,t\}$ that is only partially filled with production, then a period in this bucket is chosen to produce d_t.
c) If all buckets in $\{1,\ldots,t\}$ are either empty or full, then a largest empty available bucket is chosen to produce d_t. In this case the value of the left-hand side is incremented by one (a bucket is called available if it is empty and if its first period is in $\{1,\ldots,t\}$).

Ties are broken as follows. The production in a partially filled bucket takes place in the earliest periods. If more than one empty maximal bucket is available, then the earliest is taken.

7.2.10. Theorem.

The greedy algorithm determines the minimum number of buckets that must be used for production.

Proof. The proof is by induction on the number of periods. We use the following induction hypothesis. There is an optimal solution (the production periods of this solution are denoted by S) that produces in the same periods as the solution generated by the greedy algorithm after the demands of the periods $\{1,\ldots,t\}$ are placed. The set of production periods of the latter solution is denoted by $S_g(t)$. Thus, the induction hypothesis is $S_g(t)\subseteq S$.

If $d_{t+1}=0$, then the hypothesis also holds for S and $S_g(t+1)=S_g(t)$. Therefore, we suppose that $d_{t+1}=1$.

If there is a hole in $\{1,\ldots,t+1\}$ not in $S_g(t)$, then this hole is chosen as a production period in $S_g(t+1)$ by the greedy algorithm. If this hole is not used by S, then we can take any production period τ in S not used by $S_g(t+1)$, and transfer its production to the hole. This ensures that $S_g(t+1)\subseteq S$.

If there is no hole available, but if there is a bucket partially used for production, then the greedy algorithm takes a period in this bucket for production. Like in the previous case, we can take any production period in $S\backslash S_g(t+1)$ to transfer production to the period chosen by the greedy algorithm.

If there is no hole or partially used bucket available, then the greedy algorithm takes a largest empty available bucket, say $B_k=\{t_k,\ldots,u_k\}$. Thus, $t_k\leq t$. If S uses this bucket also, then the hypothesis holds for $S_g(t+1)$. So, suppose that S does not contain any periods of B_k. Since S uses all the buckets and holes of $S_g(t)$, there must be some period τ in $\{1,\ldots,t+1\}$ that is in S but not in $S_g(t)$. Since there are no holes in $\{1,\ldots,t+1\}\backslash S_g(t)$ τ is in a bucket B_l. The size of this bucket is at most $|B_k|$, since the greedy algorithm takes the largest bucket. We create a new solution S' from S by transferring the production from B_l to B_k. This solution uses the same number of buckets as S, and $S_g(t+1)\subseteq S'$. So, it remains to be shown that S' is feasible, i.e., that $|S'\cap N_u|\geq d_{1u}$ for $u=1,\ldots,T$. By construction of $S_g(t+1)\subseteq S'$ this holds for $u\leq t+1$.

Moreover, the production in B_k can be arranged such that the earliest periods in B_k produce. Suppose that s is the last production period in B_k. Then $|S' \cap N_u| = |S \cap N_u| \geq d_{1u}$ for all $u > s$. Now, if $|S' \cap N_u| < d_{1u}$ for some $u \in \{t+2, \ldots, s\}$, then this also holds for $u-1$, since $d_u \leq 1$ and $x_u = 1$. Thus, it holds for all earlier periods in B_k, including $t+1$, a contradiction. This proves that S' is feasible, thereby ending the correctness proof of the greedy algoritm.

□

The greedy algorithm and the solution it produces are important tools in establishing necessary and sufficient conditions for the determination of facet-defining hole/bucket-inequalities. Especially, variations in the left-hand side of a hole/bucket-inequality are used, like the following. If the variable x_t of a hole t is added to the left-hand side of a hole/bucket-inequality $\alpha x + \beta z \geq \gamma$, then γ remains equal, otherwise all optimal solutions to $\alpha x + \beta z \geq \gamma$ have production in t, i.e., they satisfy $x_t = 1$. If the variable z_u, where u is the last period of a bucket is deleted from $\alpha x + \beta z \geq \gamma$, then γ is decreased by one, otherwise all optimal solutions to $\alpha x + \beta z \geq \gamma$ would satisfy $z_u = 0$. These two observations lead to the following necessary conditions. Note that a maximal bucket is a bucket that has no larger preceding bucket.

N1) Let h be the first hole and $B = \{t, \ldots, u\}$ be the first bucket of size bigger than one. For each τ with $h \leq \tau < t$, the number of holes in $\{h, \ldots, \tau\}$ exceeds the total demand $d_{h\tau}$.

N2) Suppose that for $d_t = 1$ a production period is determined with the greedy algorithm.

   a) If there is a partially filled bucket and no empty hole, then an empty maximal bucket should be available.

   b) If all buckets are completely filled and no empty buckets are available, then an empty maximal bucket should be completely available in $\{1, \ldots, t\}$.

N3) For d_T the greedy algorithm chooses a new empty (maximal) bucket.

N4) If there are at least two buckets and $B_1, \ldots, B_k$ are of the same size, then $d_{1,u_k} \leq u_k - |B_1|$, i.e., the first bucket may be left empty.

Proof. The necessity of these conditions is proved first. Suppose that for some period N1 does not hold. Then take the minimal period τ, i.e., $d_{h\tau}$ is the number of holes in $\{h,\ldots,\tau\}$. By the minimality of τ, there are holes available for all demand periods in $\{h,\ldots,\tau\}$. Filling the holes is cheapest, since there are only buckets of size one besides the holes. However, this implies that all the holes are filled by any optimal solution, and thus the inequality is not a facet.

Suppose that for d_τ there is a partially filled bucket B available, but there is no empty maximal bucket, i.e., N2a is violated. Let s be the first period for which production is chosen to take place in B. Suppose that the last period of B is turned into a hole. If s precedes the hole, then B may be chosen, since it is still maximal. Otherwise, d_s is produced in that hole. However, for the succeeding demand period (which is period τ or an earlier period) B is still a maximal available period and may therefore be chosen by the greedy algorithm. Thus, in all cases the original greedy solution remains optimal, implying that the inequality is not a facet and that N2a must be satisfied.

Suppose that for d_τ a new bucket must be chosen by the greedy algorithm and no maximal bucket is completely available in $\{1,\ldots,\tau\}$. If no maximal bucket is available at all, then N2a is violated for the periods that have their demand produced in the last filled maximal bucket. Thus, we may assume that there is only one empty maximal bucket B, which is partially in $\{1,\ldots,\tau\}$. Thus, turning the last period of B into a hole leaves B the largest empty available bucket for τ, and therefore it can still be chosen by the greedy algorithm. Thus, the original greedy solution remains optimal, implying that the inequality is not a facet and that N2b must be satisfied.

Suppose that there is a hole h available for d_T. Let τ be the last period for which demand is not placed in a hole. Then, for all later periods $\{\tau+1,\ldots,T\}$ there are holes available in $\{\tau+1,\ldots,T\}$. Thus, any optimal solution will use these holes. Therefore, if $T\in X$, then $x_T=0$ for any optimal solution, and if $T\in Z$, then $z_T=0$ for any optimal solution. Next, suppose that there is a partially filled bucket available for d_T. Then turning the last period of this

bucket into a hole has no influence on the optimal value of the left-hand side of the inequality. We may conclude that an empty bucket must be chosen by the greedy algorithm for d_T.

Finally, if N4 does not hold, then the first bucket B_1 must be filled. Thus, then all optimal solutionssatisfy $x_{t_1}+\ldots+z_{u_1}=1$, and since the inequality contains more than one bucket, it can not be a facet.

□

Conditions N1-N4 are also sufficient. This is shown by constructing optimal solutions with properties that allow the construction of $2T$-1 linearly independent directions. These optimal solutions are characterized in S1-S4.

S1) For each bucket $B=\{t,\ldots,u\}$ there is an optimal solution with $z_u=x_u=1$, i.e., period u is the only production period in B.

S2) There is a bucket A such that
   a) For any bucket B earlier than A there is an optimal solution such that B is empty and there is one production period in A.
   b) For any bucket B later than A, there is an earlier bucket C, such that C is empty and there is one production period in B.

S3) For each hole h there is an optimal solution that leaves h unused for production.

S4) For each bucket of size one, there is an optimal solution that does not produce in this bucket.

Proof. If the greedy algorithm fills the buckets $C_1,\ldots,C_L$ in this order, then this is the order of appearance of these buckets on the planning horizon. Note that from a set of empty maximal buckets the first is chosen.

It follows directly that S1 holds for all buckets left empty and for C_L, by N3. Moreover, these solutions are such that $C_1,\ldots,C_{L-1}$ are completely filled with production. Now suppose that S1 holds for $C_{l+1},\ldots,C_L$, and let S_{l+1} be the solution with production in the last period of C_{l+1}, such that only

maximal buckets are used and such that $C_1,\ldots,C_l$ are filled with production, i.e., the filling up to C_l is in accordance with the greedy algorithm. At the moment C_l is filled with its first unit of production it is completely availabe by N2b. Thus, we take the last period of C_l for this production. At the succeeding demand period another maximal bucket C is available by N2a. This may be C_{l+1} or an earlier maximal bucket, but its size is at least $|C_l|$. Thus, this and all other demands can be transferred to C, leaving at least one period available for the production of the unit in the last period of C_{l+1}. The rest of the solution remains equal to S_{l+1}. By induction, S1 has been proved now. Note that this solution denoted by S_l only uses maximal buckets, since S_{l+1} does so. This is important in the proof of S2, which follows.

Suppose B_k is the first bucket left empty by the greedy algorithm. Then $|B_1| = \ldots = |B_{k-1}| \geq |B_k|$. Clearly, $B_1,\ldots,B_{k-1}$ have non-decreasing size, since they are all filled by the greedy algorithm. Suppose that there is some $B_i = \{t_i,\ldots,u_i\}$ $(i<k)$, that is bigger than its direct predecessor. By N2 all demands from t_i and later periods are placed in buckets of size bigger than the size of B_1. Thus, the demands of $\{1,\ldots,u_{i-1}\}$ are enough to force all the buckets $B_1,\ldots,B_{i-1}$ to be filled, in contradiction with N4. Therefore, all buckets before B_k are of the same size. By the same reason, B_k is not bigger than B_{k-1}.

If $|B_k| < |B_{k-1}|$, then we take $A = B_k$. In the solution, where B_{k-1} is filled with one unit, B_k is left empty, since it is not maximal. By N4 we know that one unit of the production in B_1 can be transferred to the first period of B_k and the rest to B_{k-1}. This gives the desired solution for B_1. Since the production of each B_i can be transferred to B_1, the solution for B_i is also available. Finally, consider the solution where a bucket B is filled with one unit. In this solution B_k is empty, and therefore the solutions in S2 can be created.

If $|B_k| = |B_{k-1}|$, then we take $A = B_{k+1}$. Consider the solution, where B_{k+1} is filled with one unit of production. Then $B_1,\ldots,B_{k-1}$ are filled and B_k is empty, since they are filled according to the greedy algorithm. By N4, we may transfer the production of B_1 to B_k. Thus, we can generate the desired solutions for $B_1,\ldots,B_k$. For buckets after B_{k+1} the filling with one unit leaves B_k empty. This proves S2.

The proof of S3 is easier. Let B be the first bucket of size bigger than one. Consider the optimal solution where B is filled with one unit of production. If the first hole h is later than B, then we can transfer production from h to B. Otherwise, if h is before B, then its production can be moved to B by N1. It follows that the first hole can be left empty. Clearly, this implies that later holes can be left empty.

For the correctness of S4 it suffices to take the solution in which the first bucket is left empty. We can move production from any bucket of size one to the first bucket without losing optimality, and making the bucket of size one empty.

□

It is now easy to construct the following directions.

Case 1. $z_\tau = 1$ if $\tau \notin Z$.

The direction $z_\tau = 1$ is created as follows. If τ is a hole, then we take the solution in which the hole is empty as a second solution. In the first we take $z_\tau = 1$. If τ is not a hole, then let it be a period in bucket $B = \{t, \ldots, u\}$. Note that $\tau = t$. If $t = u$, then B is a bucket of size one. From S4 there exists a solution that does not use B for production. With this solution the direction $z_\tau = 1$ can be created easily. Therefore, suppose that $t < u$, and consider the optimal solution that uses u as the only production period in B. From this solution the direction $z_\tau = 1$ can be created easily, also.

Case 2. $z_\tau = x_\tau = 1$, $x_t = -1$ $\tau \in Z$, where $B = \{t, \ldots, u\}$ contains τ.

The direction $z_\tau = x_\tau = 1$, $x_t = -1$ is created by considering the optimal solution where $z_u = x_u = 1$. The production in u is transferred to period τ in the first solution, necessary to create the direction, and $z_t = 1$ is added. The second direction is created from this optimal solution by transferring the production from τ to t. These solutions provide the suggested direction.

Case 3. $x_\tau = 1$, if $\tau \notin X$.

If τ is a hole, then this direction can be created directly from the solution in case 1. Otherwise, τ is in a bucket $B = \{t, \ldots, u\}$ $(\tau \neq t)$. The direction $x_\tau = 1$

is created by considering the optimal solution where $z_{\tau-1}=x_{\tau-1}=1$. This is the second solution. The first solution is created from the second one, by adding $x_\tau=1$. These solutions provide the suggested direction.

Case 4. $x_\tau=1$, $x_t=-1$, if $\tau\in X$ for some choice of $t\in X$.
The main problem here is to ensure that these directions are linearly independent. This is reached by making the following choices. Consider A as defined in S2. Let $\tau\in B$, if B precedes A, then t is chosen as the first period of A. Otherwise, t is chosen as the first period of an empty bucket preceding B, which exists according to S2. Note that for A itself no direction is created.

The number of created directions is now: $(T-|Z|)+|Z|+(T-|X|)+(|X|-1)=2T-1$. This is the desired number of directions. The proof that these directions are linearly independent is left to the reader.

A final remark concerning these hole/bucket-inequalities concerns their number. There are exponentionally many such facets, as follows by the following example. Suppose that on the planning horizon $\{1,\ldots,T\}$ $(T\geq 5)$, we have the following demand pattern: $d_1=\ldots=d_{T-3}=0$, $d_{T-2}=d_{T-1}=d_T=1$. Then any inequality with two or more buckets of size two and additional buckets of size one defines a facet. There are exponentially many inequalities with this structure.

7.2.4. Some other classes of facets with 0/1-coefficients

In the following, we allow periods in the intersection of X and Z. As for the holes, there are necessary conditions that must be met by an inequality containing such periods, in order to be a facet. In the case of a period $t\in X\cap Z$, an obvious condition is that there must be an optimal solution which satisfies $z_t=1$. This condition can be strengthened a little more, by considering a maximal batch of consecutive periods all satisfying membership of $X\cap Z$. If t is the first period of the batch, the condition is met when t satisfies it. Then for any period u in the batch $z_u=1$ is realized by shifting the production of the periods $\{t,\ldots,u-1\}$ one unit to the left.

The following four types of periods may be available in the left-hand side of an inequality.

1) $t \notin X \cup Z$; 2) $t \in X \setminus Z$; 3) $t \in Z \setminus X$; 4) $t \in X \cap Z$.

These periods will be denoted by the values of the coefficients of the variables x_t and z_t, respectively. Thus, the periods of class 1 are called 00-periods, the periods of class 2 are called 10-periods, the periods of class 3 are called 01-periods, and the periods of class 4 are called 11-periods.

These periods cannot appear in any arbitrary order in facets. The following lemma provides some forbidden orders of the periods.

7.2.11. Lemma.

If t is a 00-period, then $t+1 \notin Z$.

If $t \in X$ and $t+1 \notin Z$, then $t \notin Z$.

Proof. Suppose that t is a 00-period, and that $t+1 \in Z$. Consider a solution in which $z_{t+1}=1$. This solution can be improved by setting $x_t = z_t = 1$, and $z_{t+1}=0$.

Suppose that $t \in X \cap Z$ and $t+1 \notin Z$. Then a solution in which $z_t = 1$ can be improved by setting $z_{t+1}=1$, and $z_t = 0$. This reduces the cost by one unit. The possible production in period t can be moved to the earliest non-production period, without increasing the cost of the solution, since no start-up costs have te be incurred. This is true because either the preceding period is a production period or the period itself is period 1 and $1 \notin Z$.

□

Put briefly, lemma 7.2.11 states the following. A batch of 00-periods is followed by a 10-period. A batch of 11-periods can be followed by 01-periods only. Finally, 10-periods and 01-periods can be followed by all types of periods. The only restriction here concerns 01-periods when followed by 11-periods. This situation is dealt with in the following lemma.

7.2.12. Lemma.

If $t \in Z \setminus X$ *and* $t+1 \in Z \cap X$, *then* $t-1 \in Z \cap X$.

Proof. Suppose that $t \in Z \setminus X$, $t+1 \in Z \cap X$, and $t-1 \notin Z \cap X$. Consider a solution in which $z_{t+1} = 1$. Let t be the first period of a production batch consisting of the periods $\{t+1, \ldots, u\}$. If all these periods are in X, then a cost reduction can be achieved by shifting the batch one period earlier. Thus, suppose that τ is the first period in the batch not in X. Then a new solution is created, by setting $z_\tau = 1$, and shifting the production of the periods $\{t+1, \ldots, \tau-1\}$ two periods to the left. This results in a cost reduction.

□

In the following we will construct several classes of facets which have a specific structural property. The first class is a set of facets that contain a 01-period, surrounded by 11-periods. The second class contains facets, where a batch of $p_1 \geq 1$ 11-periods is succeeded by a batch of $p_2 \geq 1$ 01-periods. This structure is called a shift. The third set consists of shifts with $p_1 \geq 1$ and $p_2 = 1$. There is no doubt that these classes can be extended, by relaxing the conditions under which they are facets. We will give these classes without proofs. The reader who wishes to create the directions should create optimal solutions, such that the first period of a batch of 11-periods is a start-up period.

CLASS 1. Here a structure exists consisting of one period in $X \setminus Z$, followed by p_1+1 periods, from which the first p_1 are in $X \cap Z$ and the last is in $Z \setminus X$. Then an arbitrary (positive) number of blocks of p_2+1 $(p_1 \geq p_2)$ periods, where the first p_2 are in $X \cap Z$ and the last is in $Z \setminus X$, follows. The number of demand periods in the first block is $p_1 - p_2$, and the number of demand periods starting from the first period of the last p_2 block is equal to p_2+1.

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
d_t	0	0	0	0	0	0	1	0	0	0	0	1	1	1

$$\begin{aligned} & z_5+z_6+z_7+z_8+z_9+z_{10}+z_{11}+z_{12}+z_{13}+z_{14} \\ x_1+x_2+x_3+x_4+x_5+x_6+x_7+ \quad & x_9+x_{10}+ \quad x_{12}+x_{13} \end{aligned} \geq 4$$

This type of inequalities can be generalized in several ways, for instance, by adding holes before the structure.

CLASS 2. The second class consists of one structure of the following type. One period is in $X\backslash Z$, the following $p_1 \geq 1$ periods are in $X \cap Z$, and the following $p_2 \geq 1$ periods are in $Z\backslash X$. This structure is called a (p_1,p_2)-shift. The p_1 periods are demand periods, and the p_2 periods not except the last one. The first period of the structure is also not a demand period.

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
d_t	0	0	0	0	1	1	1	1	1	1	0	0	0	1

$$\begin{array}{rr} & z_5+z_6+z_7+z_8+z_9+z_{10}+z_{11}+z_{12}+z_{13}+z_{14} \\ x_1+x_2+x_3+x_4+x_5+x_6+x_7+x_8+x_9+x_{10}+ & \end{array} \geq 6$$

This type of structure can be generalized by adding holes before the structure, and by adding a complete hole/bucket-inequality behind the structure.

CLASS 3. This type of structure contains an arbitrary amount of $(p_1,1)$-shifts, where shifts with different values for p_1 may occur in the same inequality. Unfortunately, it is hard to determine sufficient conditions for the demand function, to create a facet. Hopefully, the examples give some idea of what these conditions should be.

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
d_t	0	0	0	0	0	0	1	1	1	0	1	1	1	1

$$\begin{array}{rrr} & z_3+z_4+z_5+ & z_8+z_9+z_{10}+ \quad z_{12}+z_{13}+z_{14} \\ x_1+x_2+x_3+x_4+ & x_6+x_7+x_8+x_9+ & x_{11}+x_{12}+x_{13} \end{array} \geq 6$$

$$\begin{array}{rrr} & z_5+z_6+z_7+z_8+ & z_{13}+z_{14} \\ x_1+x_2+x_3+x_4+x_5+x_6+x_7+ & x_9+x_{10}+x_{11}+x_{12}+x_{13} & \end{array} \geq 6$$

These inequalities can be generalized by adding holes.

This subsection is ended with some examples that do not belong to the three classes mentioned above. The first one has a bucket of size $n>2$ before a $(1,n)$-shift. The second has a $(1,2)$-shift before a $(2,1)$-shift.

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
d_t	0	0	0	0	0	0	0	0	0	1	1	1	1	1

$$x_1+x_2+z_3+z_4+z_5+x_6+x_7+x_8+x_9+z_9+z_{10}+z_{11}+z_{12}+z_{13}+x_{14} \geq 2$$

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
d_t	0	0	1	0	0	0	0	0	1	1	1	1	1	1

$$x_1+x_2+x_3+z_3+z_4+z_5+x_6+x_7+x_8+x_9+x_{10}+z_9+z_{10}+z_{11}+x_{12}+x_{13}+x_{14} \geq 5$$

A final remark concerns those facet-defining inequalities that are also facet defining for the original polyhedron. Without proof we state that if the periods $t-1,\ldots,T-1$ are no demand periods, then the z_t variable must be replaced by x_t. This is due to the fact that in the original polyhedron the inequality $z_t \geq x_t$ is a valid one.

7.3. An uncapacitated facility location formulation of DLS

The difficulties with respect to a polyhedral description of DLS have become clear in the preceding section. A facility location formulation can be obtained by splitting the production variables of each period to specify for which demand production is incurred. Also, the start-up variables must be split. We will derive a min cost flow model for the facility location formulation, which yields a complete linear description.

The demand periods are numbered in increasing order as follows: $t_1,t_2,\ldots,t_D$, where D is the cumulative demand of the periods $\{1,\ldots,T\}$. The start-up

variables are split as well as the production variables, according to the period for which is produced as follows. For $t\in\{1,\ldots,T\}$ and $i\in\{1,\ldots,D\}$:

$$x_{t,i}=\begin{cases}1 \text{ if there is production in period } t \text{ for period } t_i;\\ 0 \text{ otherwise.}\end{cases}$$

$$z_{t,i}=\begin{cases}1 \text{ if } x_{t,i}=1 \text{ and } x_{t-1,i-1}=0;\\ 0 \text{ otherwise.}\end{cases}$$

This variable splitting leads to the following model.

$$\text{(DLS-UFL)} \quad \min \sum_{i=1}^{D}\sum_{t=1}^{t_i}(g_t z_{t,i}+c_t x_{t,i}) \tag{7.3.1}$$

$$\sum_{t=1}^{t_i} x_{t,i}=1 \qquad (1\le i\le D) \tag{7.3.2}$$

$$z_{t,i}+x_{t-1,i-1}\ge x_{t,i} \qquad (1\le t\le T;1\le i\le D) \tag{7.3.3}$$

$$x_{t,i}\in\{0,1\} \qquad (1\le t\le T;1\le i\le D) \tag{7.3.4}$$

$$z_{t,i}\in\{0,1\} \qquad (1\le t\le T;1\le i\le D) \tag{7.3.5}$$

DLS-UFL can be modelled as a shortest path problem on a network, defined as follows. Note that a shortest path problem between two vertices is a min cost flow problem.

For each $i\in\{1,\ldots,D+1\}$, and for each t: $i\le t\le t_i$ ($t_{D+1}\equiv T+1$), two vertices $u_{t,i}$ and $v_{t,i}$, are defined. A shortest path is to be found between the vertices $u_{1,1}$ and $u_{T+1,D+1}$. The arcs are defined as follows.

$(u_{t,i},v_{t,i})$ $(1\le i\le D+1;i\le t\le t_i)$ with capacity 1, cost g_t, and flow $z_{t,i}$;
$(v_{t,i},u_{t,i})$ $(1\le i\le D+1;i\le t\le t_i)$ with capacity 2, cost 0, and flow $\beta_{t,i}$;
$(u_{t,i},u_{t,i+1})$ $(1\le i\le D+1;i\le t<t_i)$ with capacity 1, cost 0, and flow $\alpha_{t,i}$;
$(v_{t,i},v_{t+1,i+1})$ $(1\le i<D+1;i\le t<t_i)$ with capacity 1, cost c_t, and flow $x_{t,i}$.

The optimal path from $u_{1,1}$ to $u_{T+1,D+1}$ may not be simple, if the start-up costs are allowed to be negative. Therefore, the capacity of the arcs $(v_{t,i},u_{t,i})$ is set to 2.

EXAMPLE: $T = 6$

t	1	2	3	4	5	6
d_t	0	1	0	0	1	1

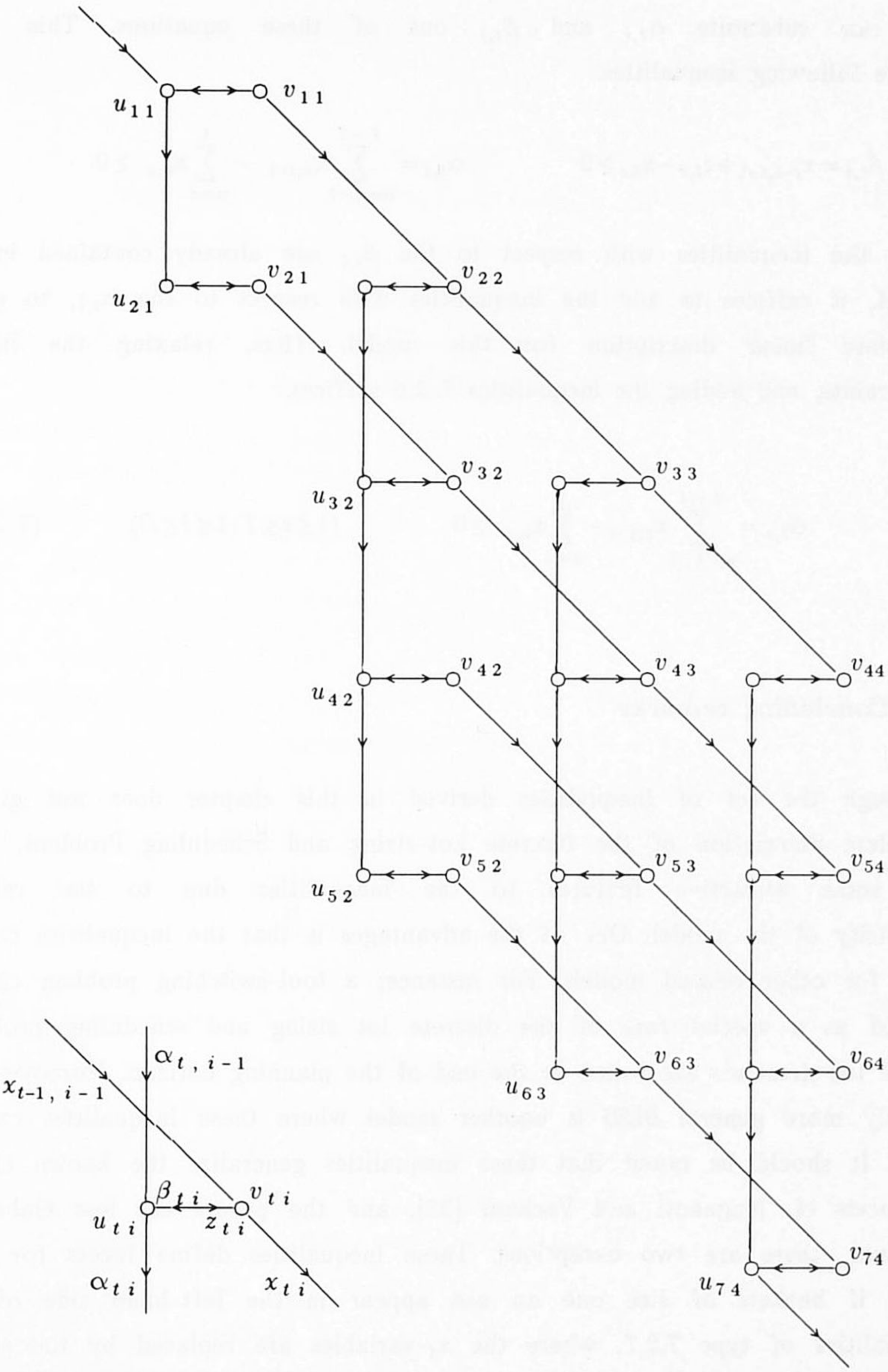


Figure 7.1. Min cost flow network.

From the flow conservation constraints in each vertex we get:

$$u_{t,i}: \quad \alpha_{t,i-1}+\beta_{t,i}=\alpha_{t,i}+z_{t,i} \qquad\qquad v_{t,i}: \quad x_{t-1,i-1}+z_{t,i}=x_{t,i}+\beta_{t,i}$$

We can substitute $\alpha_{t,i}$ and $\beta_{t,i}$ out of these equations. This leads to the following inequalities.

$$\beta_{t,i}=x_{t-1,i-1}+z_{t,i}-x_{t,i}\geq 0 \qquad\qquad \alpha_{t,i}=\sum_{u=i-1}^{t-1} x_{u,i-1} - \sum_{u=i}^{t} x_{u,i} \geq 0$$

Since the inequalities with respect to the $\beta_{t,i}$ are already contained in the model, it suffices to add the inequalities with respect to the $\alpha_{t,i}$, to get a complete linear description for this model. Thus, relaxing the integer constraints, and adding the inequalities 7.3.6 suffices.

$$\alpha_{t,i}=\sum_{u=i-1}^{t-1} x_{u,i-1} - \sum_{u=i}^{t} x_{u,i} \geq 0 \qquad (1\leq t\leq T;1\leq i\leq D) \qquad (7.3.6)$$

7.4. Concluding remarks

Although the set of inequalities derived in this chapter does not give a complete description of the Discrete Lot-sizing and Scheduling Problem, there are some attractive features to the inequalities due to the relative simplicity of the model. One of the advantages is that the inequalities can be used for other related models. For instance, a tool-switching problem can be viewed as a special case of the discrete lot sizing and scheduling problem, where the demands are found at the end of the planning horizon. Moreover, the slightly more general DLSS is another model where these inequalities can be used. It should be noted that these inequalities generalize the known classes of facets cf. Magnanti and Vachani [32], and the proofs are less elaborate. However, there are two exceptions. These inequalities define facets for DLSS only, if buckets of size one do not appear in the left-hand side of the inequalities of type 7.2.7, where the x_t-variables are replaced by the set-up variables y_t $(1\leq t\leq T)$. Another notable exception is provided by Sastry [46],

who describes a class of inequalities with coefficients other than -1,0,1, and these are therefore not contained in the classes of section 7.2.

REFERENCES

[1] Aggarwal, A., and J.K. Park (1990), Improved algorithms for economic lot-size problems, *Working paper, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge.*

[2] Aho, A.V., J.E. Hopcroft and J.D. Ullmann (1983), Data structures and algorithms, *Addison–Wesley series in Computer Science and Information Processing.*

[3] Barany, I., T.J. Van Roy and L.A. Wolsey (1984), Strong formulations for multi-item capacitated lot-sizing, *Management Science* **30** (10), pp. 1255–1261.

[4] Barany, I., T.J. Van Roy and L.A. Wolsey (1984), Uncapacitated lot-sizing: the convex hull of solutions", *Mathematical Programming Study* **22**, pp. 32–43.

[5] Bellman, R. (1957), Dynamic Programming, *Princeton University Press, Princeton, New Jersey.*

[6] Bitran, G.R., T.L. Magnanti and H.H. Yanasse (1984), Approximation methods for the uncapacitated dynamic lot size problem, *Management Science* **30**, pp. 1121–1140.

[7] Bitran, G.R., and H.H. Yanasse (1982), Computational complexity of the capacitated lot size problem, *Management Science* **28**, pp. 1174–1186.

[8] Denardo, E.V. (1982), Dynamic programming: models and applications, *Prentice–Hall.*

[9] Edmonds, J. (1965), Paths, trees, and flowers, *Canadian Journal of Mathematics* **17**, pp.449–467.

[10] Eppen, G.D., F.J. Gould and B.P. Pashigian (1969), Extensions of the planning horizon theorem in the dynamic lot size problem, *Management Science* **5**, pp. 268–277.

[11] Eppen, G.D. and R.K. Martin (1988), Solving multi-item capacitated lot-sizing problems using variable redefinition, *Operations Research* **35** (6), pp. 832–848.

[12] Erlenkotter, D. (1990), Ford-Whitman Harris and the economic order

quantity model, *Operations Research* **38**, pp. 937–946.

[13] Evans, J.R. (1985), An efficient implementation of the Wagner–Whitin algorithm for dynamic lot–sizing, *Journal of Operations Management* **5**, pp. 229–235.

[14] Fleischmann, B. (1988), "The discrete lot–sizing and scheduling problem". *Technical Report*, *Institut fur Unternehmensforschung*, *Universitat Hamburg*.

[15] Florian, M., J.K. Lenstra and A.H.G. Rinnooy Kan (1980), Deterministic production planning with concave costs and capacity constraints, *Management Science* **26**, pp. 12–20.

[16] Gomory, R.E. (1958), Outline of an algorithm for integer solutions to linear programs, *Bulletin of the American Mathematical Society* **64**, pp. 275–278.

[17] Grötschel, M., L. Lovász and A. Schrijver (1981), The ellipsoid method and its uses in combinatorial optimization, *Combinatorica* **1**, pp. 169–197.

[18] Harris, F.W. (1913), How many parts to make at once, *Factory*, *the Magazine of Management* **10**, pp. 135–136, 152.

[19] Hoesel, C.P.M. van (1991), An $O(n\log n)$ algorithm for the two–machine flow shop problem with controllable machine speeds, *Technical Report* 9112/*A*, *Econometric Institute*, *Erasmus University Rotterdam*, *the Netherlands*.

[20] Hoesel, C.P.M. van, A.W.J. Kolen and A.P.M. Wagelmans (1989), A dual algorithm for the economic lot sizing problem, *Report* 8940/*A*, *Econometric Institute*, *Erasmus University Rotterdam*, *the Netherlands*.

[21] Hoesel, C.P.M. van, A.W.J. Kolen, A.H.G. Rinooy Kan and A.P.M. Wagelmans (1989), Sensitivity analysis in combinatorial optimization: a bibliography, *Report* 8944/*A*, *Econometric Institute*, *Erasmus University Rotterdam*, *the Netherlands*.

[22] Hoesel, C.P.M. van, and A.P.M. Wagelmans (1989), A note on "stability of the constant cost dynamic lot size model" by K. Richter, *Report* 8959/*A*, *Econometric Institute*, *Erasmus University Rotterdam*, *the Netherlands*.

[23] Karmarkar, U.S., and L. Schrage (1985), The deterministic dynamic cycling problem, *Operations Research* **33**, pp. 326–345.

[24] Karmarkar, U.S., S. Kekre and S. Kekre (1987), The dynamic lot–sizing

problem with startup and reservation costs, *Operations Research* **35**, pp. 389–398.

[25] Khachian, L.G. (1979), A polynomial algorithm in linear programming, *Soviet Mathematics Doklady* **20**, pp. 191–194.

[26] Klawe, M. (1990), A simple linear time algorithm for concave one-dimensional dynamic programming, *Preprint*, *University of British Columbia*, *Vanvouver*, *Canada*.

[27] Krarup ,J., and O. Bilde (1977), Plant location, set covering, and economic lot-size: An $O(mn)$ algorithm for structured problems, in *Numerische Methoden bei Optimierungsaufgaben*, *Band* 3: *Optimierung bei Graphentheorietischen und Ganzzahligen Problemen*, *Birkhauser*, pp. 155–186.

[28] Kuik R., M. Salomon, S. van Hoesel and L.N. van Wassenhove (1989), The single item discrete lotsizing and scheduling problem: linear description and optimization, *Management report series no.* 53, *Erasmus University*, *Rotterdam*, *the Netherlands*.

[29] Lee, I.-S. (1986), On sensitivity analysis for shortest path dynamic programming models, *Working Paper*, *University of California*, *Los Angeles*, *USA*.

[30] Lovász, L. (1979), Graph theory and integer programming, *Annals of Discrete Mathematics* **4**, pp. 141–158.

[31] Lundin, R.A. and T.E. Morton (1975), Planning horizons for the dynamic lot size model: Zabel vs. protenctive procedures and computational results, *Operations Research* **23**, pp. 711–734.

[32] Magnanti, T.L., and R. Vachani, A strong cutting plane algorithm for production scheduling with changeover costs, *Working Paper* (*OR* 173–87), *Massachusetts Institute of Technology*, *Cambridge*, *USA*.

[33] Manne, A.S (1958), Programming of economic lot sizes, *Management Science* **4**, pp. 115–135.

[34] Martin, R.K., R.L. Rardin and B.A. Campbell, Polyhedral characterization of discrete dynamic programming, *Working paper*, *CC*–87–24, *Purdue University*, *West Lafayette*, *Indiana*, *USA*.

[35] Minkowski, H. (1896), Geometrie der Zahlen (erste lieferung), *Teubner*, *Leipzig*, 1986.

[36] Motzkin. T.S. (1936), Beiträge zur Theorie der linearen Ungleichungen

(Inaugural dissertation Basel), *Azriel*, *Jerusalem*, 1936.

[37] Nemhauser, G.L., and L.A. Wolsey (1988), Integer and combinatorial optimization, *Wiley*, *Interscience Series in Discrete Mathematics and Optimization.*

[38] Pochet, Y. (1988), Valid inequalities and separation for capacitated economic lot sizing, *Operations Research Letters* **7**, pp. 109–116,

[39] Pochet, Y., and L.A. Wolsey (1988), Lot–sizing models with backlogging: strong reformulations and cutting planes, *Mathematical Programming* **40**, pp. 317–335.

[40] Preparata, F.P., and M.I. Shamos (1985), Computational Geometry, an introduction, *Springer*, *New York.*

[41] Rardin, R.L., and L.A. Wolsey (1990), Valid inequalities and projecting the multicommodity extended formulation for uncapacitated fixed charge network flow problems, *Discussion paper no.* 9024, *Center for Operations Research and Econometrics*, *University of Louvain*, *Louvain–La–Neuve*, *Belgium.*

[42] Richter, K. (1987), Stability of the constant cost dynamic lot size model, *European Journal of Operational Research* **31**, pp. 61–65.

[43] Richter ,K., and J. Vörös (1989), A parametric analysis of the dynamic lot–sizing problem, Journal of Information Processing Cybernetics EIK **25**, pp. 67–73.

[44] Richter ,K., and J. Vörös (1989), On the stability region for multi–level inventory problems, *European Journal of Operational Research* **41**, pp. 169–173.

[45] Salomon ,M. (1990), Deterministic lot sizing models for production planning, *Ph.D. thesis*, *Erasmus University*, *Rotterdam*, *the Netherlands.*

[46] Sastry, T. (1990), Polyhedral structure of the product cycling problem with changeover costs, *Ph.D. Thesis*, *Massachusetts Institute of Technology*, *Cambridge*, *USA.*

[47] Saydam, C., and M. McKnew (1987), A fast microcomputer program for ordering using the Wagner–Whitin algorithm, *Production and Inventory management Journal* **28**, pp. 15–19.

[48] Schrage, L. (1982), The multiproduct lot scheduling problem, in *Deterministic and Stochastic Scheduling*, *M.A.H. Dempster*, *J.K. Lenstra and A.H.G. Rinnooy Kan*, *eds.*, *Nato advanced Study Institutes series*, *D.*

Reidel, Holland.

[49] Schrijver, A. (1986), Theory of linear and integer programming, *Wiley, Interscience Series in Discrete Mathematics.*

[50] Veinott, A.F. jr. (1964), Production planning with convex costs: a parametric study, *Management Science* **10** (no. 4), pp. 441–460.

[51] Wagelmans, A.P.M. (1990), Sensitivity analysis in combinatorial optimization, *Ph.D. thesis, Erasmus University Rotterdam, the Netherlands.*

[52] Wagelmans, A.P.M., C.P.M. Van Hoesel and A.W.J. Kolen (1989), Economic lot–sizing: An $O(n\log n)$ algorithm that runs in linear time in the Wagner–Whitin Case", *Report* 8952/*A*, *Econometric Institute, Erasmus University Rotterdam, the Netherlands.* To appear in *Operations Research.*

[53] Wagner, H.M. (1960), A postscript to dynamic problems in the theory of the firm, *Naval Research Logistics Quarterly* **7**, pp. 7–12.

[54] Wagner, H.M., and T.M. Whitin (1958), A dynamic version of the economic lot size model", *Management Science* **5**, pp. 89–96.

[55] Wassenhove, L.N. van, and P. Vanderhenst (1983), Planning production in a bottleneck department, *European Journal of Operational Research* **12**, pp. 127–137.

[56] Weyl, H. (1935), Elementare Theorie der konvexen Polyeder, *Commentarii Mathematici Helvetici* **7**, pp. 290–306.

[57] Wolsey, L.A. (1989), Uncapacitated lot–sizing problems with start–up costs", *Operations Research* **37**, pp. 741–747.

[58] Yao, F. (1982), Speed–up in dynamic programming, *SIAM Journal on Algebraic and Discrete Methods* **3**, pp. 532–540.

[59] Zabel, E. (1964), Some generalizations of an inventory planning horizon theorem, *Management Science* **10**, pp. 465–471.

[60] Zangwill, W.I. (1966), A deterministic multi–period production scheduling model with backlogging, *Management Science* **13**, pp. 105–119.

[61] Zangwill, W.I. (1969), A backlogging and a multi–echelon model of a dynamic economic lot–sizing production system: a network approach, *Management Science* **15**, pp. 506–527.

SAMENVATTING

MODELLEN EN ALGORITMEN VOOR SERIE-GROOTTE BEPALING VAN EEN PRODUKT

Het probleem voor de bepaling van de serie-grootte van een produkt op een machine kan als volgt geformuleerd worden. Een eindige tijdshorizon is verdeeld in een aantal perioden. In deze perioden treedt een vraag naar een gegeven aantal eenheden van een produkt op. Er dient een produktieschema ontworpen te worden zodanig dat aan de vraag voldaan kan worden en wel tegen minimale kosten. De kosten bestaan onder andere uit produktie- en voorraadkosten, de variabele kosten. Karakteristiek voor serie-grootte problemen zijn echter de vaste kosten. Deze kosten, de zogenaamde opzetkosten, worden uitsluitend in rekening gebracht in perioden waarin geprodukeerd wordt. Het aldus geformuleerde probleem heet het economisch serie-grootte probleem.

Er bestaan diverse uitbreidingen van en varianten op het economisch serie-grootte probleem. Een belangrijke uitbreiding is de introduktie van een tweede type vaste kosten, de zogenaamde opstartkosten. De opstartkosten worden slechts bij aanvang van produktie in rekening gebracht, ongeacht het aantal aaneengesloten perioden waarin geproduceerd wordt, terwijl de opzetkosten in iedere produktieperiode in rekening gebracht worden. Een belangrijke variant op het economisch serie-grootte probleem is het discrete serie-grootte probleem. Ten eerste bestaat de vaste kosten component bij dit probleem uit opstartkosten. Ten tweede zijn er beperkingen ten aanzien van de produktie in iedere periode. Men kan kiezen uit twee alternatieven, namelijk een vaste gegeven hoeveelheid produceren of niet produceren. Het economisch en het discrete serie-grootte probleem vormen, met diverse combinaties van de genoemde typen vaste kosten, het hoofdbestanddeel van de in dit proefschrift beschouwde problemen.

De bovengenoemde serie-grootte problemen zijn allemaal zogenaamde makkelijke problemen. Met behulp van dynamische programmering zijn deze problemen in polynomiale tijd oplosbaar. De dynamische programmeringsalgoritmen voor deze problemen zijn reeds langere tijd bekend. Een van de belangrijkste resultaten in dit proefschrift betreft de verbetering van deze algoritmen. Het blijkt dat met behulp van eenvoudige technieken uit de computermeetkunde de rekentijd van het dynamisch programmeringsalgoritme voor het economisch serie-grootte probleem sterk verbeterd kan worden. Dit geldt ook voor de varianten, waarbij opstartkosten toegevoegd zijn en/of nalevering toegestaan wordt. De algoritmen voor het discrete serie-grootte probleem en zijn varianten kunnen met soortgelijke technieken eveneens verbeterd worden.

De hierboven beschreven serie-grootte problemen kunnen als gemengd geheeltallige lineare programmeringsproblemen geformuleerd worden. In het algemeen zijn de geheeltalligheidseisen in deze modellen noodzakelijk. Echter, zoals in dit proefschrift besproken wordt, zijn voor de serie-grootte problemen lineaire beperkingen bekend, die de geheeltalligheidseisen in de formulering overbodig maken. Dit laatste is ook te realiseren door de in de formulering gebruikte variabelen voor het modelleren van produktie te splitsen. De formuleringen die gebaseerd zijn op deze splitsing van variabelen hebben verwantschap met modellen in de locatietheorie. Ze worden dan ook locatieformuleringen voor serie-grootte problemen genoemd. Voor het economisch serie-grootte probleem worden de expliciete lineaire beperkingen gegeven en er wordt aangetoond dat de geheeltalligheidseisen daarmee overbodig worden. Hetzelfde geldt voor het economisch serie-grootte probleem met opstartkosten. Voor het discrete serie-grootte probleem wordt een niet-volledige klasse lineaire beperkingen geformuleerd. De locatie formulering wordt voor alle hier genoemde problemen gegeven.

In hoofdstuk 1 worden de diverse serie-grootte problemen geformuleerd. De dynamische programmeringsalgoritmen worden geformuleerd en de theorie behorende bij de problematiek van de lineaire beperkingen wordt besproken.

In hoofdstuk 2 worden twee, voor dit proefschrift elementaire, algoritmen voor het economisch serie-grootte probleem besproken. Beide algoritmen zijn

gebaseerd op lineaire programmeringsformuleringen van het economisch serie-grootte probleem. Het eerste algoritme geeft een constructief bewijs voor het feit dat de gegeven lineaire beperkingen de geheeltalligheidseisen overbodig maken. Dit algoritme inspireerde onderzoek naar oplossingsmethoden voor het locatiemodel van het economisch serie-grootte probleem. De maximale rekentijd van algoritmen voor dit model blijkt aanmerkelijk lager dan voor tot nu toe bekende algoritmen.

In hoofdstuk 3 wordt het algoritme voor het locatiemodel uit hoofdstuk 2 geformuleerd als een dynamisch programmeringsalgoritme. Hieruit blijkt dat dit algoritme niet alleen gebruikt kan worden voor het economisch serie-grootte probleem, maar ook eenvoudig aangepast kan worden voor toepassing op generalisaties van dit probleem, zoals het serie-grootte probleem waarbij nalevering toegestaan is en het serie-grootte probleem met zowel opstart- als opzetkosten.

In hoofdstuk 4 worden dynamische programmeringsalgoritmen beschreven voor het discrete serie-grootte probleem. Deze blijken weliswaar niet alle karakteristieken van de algoritmen uit hoofdstuk 3 te bevatten, maar beide typen algoritmen zijn gebaseerd op technieken uit de computermeetkunde.

In hoofdstuk 5 wordt het dynamisch programmeringsalgoritme uit hoofdstuk 3 gebruikt om de gevoeligheid van optimale oplossingen af te leiden, voor veranderingen in de parameterwaarden voor het economische serie-grootte probleem. Dit blijkt voor nagenoeg alle soorten parameters met behulp van de ontwikkelde technieken sneller te kunnen dan voorheen.

In hoofdstuk 6 wordt een volledige lineaire beschrijving gegeven van het economisch serie-grootte probleem, waarbij opstart- en opzetkosten berekend worden. Naast de lineaire beschrijving wordt ook een scheidingsalgoritme besproken, en wordt het locatiemodel behandeld.

In hoofdstuk 7 wordt een uitgebreide, maar niet volledige lineaire beschrijving van het discrete serie-grootte probleem gegeven. Het blijkt dat er een grote variëteit aan typen lineaire beperkingen voor dit probleem is. Nu

blijkt opnieuw de kracht van de locatieformulering voor serie-grootte problemen, daar voor deze formulering een korte volledige lineaire beschrijving gerealiseerd kan worden.

CURRICULUM VITAE

Stan van Hoesel (1961) completed his secondary school education at the RSG Willem II, Tilburg, in 1979. In the same year he started his studies for mathematical engineer at the Eindhoven University of Technology. He specialized in discrete mathematics and optimization. He wrote his master's thesis in 1986 on disjoint paths in graphs under supervision of Prof.dr. A. Schrijver. In 1987 he started as a Ph.D. student at Erasmus University Rotterdam. The project "sensitivity analysis in combinatorial optimization" was sponsored, during three years, by NWO. Part of this research took place in Bonn, Germany, at the Friedrich Wilhelms University, on a grant of ERASMUS. In the beginning of 1991 he started as an assistant professor at Tilburg University. His current position is at the department of mathematics of the Eindhoven University of Technology.

STELLINGEN

behorende bij het proefschrift

Models and algorithms for single-item lot sizing problems

Constantinus Peter Maria van Hoesel

Erasmus Universiteit Rotterdam, november 1991

1

Diverse gecapaciteerde serie-grootte problemen zijn polynomiaal oplosbaar. De complexiteit van de hiervoor ontwikkelde dynamische programmeringsalgoritmen wordt vaak ad hoc bepaald (zie Bitran and Yanasse, en Florian et al. voor enige voorbeelden). Deze resultaten zijn af te leiden met één recursieformule, waarbij de berekening van de kosten van deelproblemen de complexiteit van deze algoritmen bepaalt.

Bitran, G.R., and H.H. Yanasse (1982), Computational complexity of the capacitated lot size problem, *Management Science* **28**, pp. 1174-1186.

Florian, M., J.K. Lenstra and A.H.G. Rinnooy Kan (1980), Deterministic production planning with concave costs and capacity constraints, *Management Science* **26**, pp. 12-20.

2

Het twee-machine flow shop probleem, waarbij machine-snelheden regelbaar zijn kan opgelost worden met behulp van een algoritme, dat dezelfde complexiteit heeft als het algoritme voor het probleem met vaste machine-snelheden.

Hoesel, C.P.M. van (1991), An $O(n\log n)$ algorithm for the two-machine flow shop problem with controllable machine speeds, *Technical Report* 9112/*A*, *Econometric Institute*, *Erasmus University Rotterdam*, *the Netherlands*.

3

De optimale koppelingen in een graaf, waarbij de kosten van de kanten lineair afhangen van een parameter, kunnen met behulp van het bloesem algoritme van Edmonds "on-line" berekend worden.

Hoesel, C.P.M. van, A.H.G. Rinnooy Kan, and A.P.M. Wagelmans (1990), Edmonds' blossom algorithm tailored for parametric analysis of the perfect matching problem, unpublished manuscript.

4

Het onderscheid tussen *NP*-moeilijke en makkelijke problemen is contra-intuïtief. Het suggereert lange rekentijden voor *NP*-moeilijke problemen en eenvoudige algoritmen voor makkelijke problemen. Het eerste hoeft niet waar te zijn, terwijl het tweede juist zelden waar is: de algoritmen met de kortste rekentijd zijn vaak zeer gecompliceerd.

5

Beschouw het economisch serie-grootte probleem met constante opzet- en voorraadkosten en zonder productiekosten. De optimale productieschemas voor alle positieve verhoudingen tussen opzet- en voorraadkosten kunnen berekend worden in een aantal stappen dat kwadratisch is in het aantal perioden.

Hoesel, C.P.M. van, and A.P.M. Wagelmans (1989), A note on "Stability of the constant cost dynamic lot size model by K. Richter", *Technical Report* 8959/*A*, *Econometric Institute*, *Erasmus University Rotterdam*, *the Netherlands*.

6

In een planair ingebedde graaf met één gat, waarvan de punten voldoen aan een pariteitsconditie, is de snede-conditie een noodzakelijke en voldoende voorwaarde voor het bestaan van paarsgewijs kant-disjuncte paden van een gegeven homotopie.

Hoesel, C.P.M. van, and A. Schrijver (1990), Edge-disjoint homotopic paths in a planar graph with one hole, *Journal of Combinatorial Theory* **48**, *series B*.

7

Praktijk en theorie zijn duale begrippen. Een van de meest eminente taken van de Combinatorische Optimalisering is het verkleinen of zelfs het dichten van dualiteitskloven.

8

Het probleem waarbij aan de vraag naar telefoonlijnen in een telecommunicatie netwerk moet worden voldaan, waarbij de multiplexers reeds in het netwerk geplaatst zijn, is NP-volledig.

Bikker, F., V. Dijk and C.P.M. van Hoesel, Minimization of multiplexers in a telecommunications network, to appear.

9

Verbeteringen van de looptijd van algoritmen kunnen op twee manieren verwezenlijkt worden: enerzijds door betere datastructuren toe te passen, anderzijds door de structuur van het onderhavige probleem beter te doorgronden. Aandacht voor beide methodieken strekt tot voordeel van de onderzoeker.

10

Als het antwoord op een van de meest intrigerende vragen van de combinatorische optimalisering, te weten: is P gelijk aan NP, gevonden wordt, dan zal het vakgebied aan belangstelling winnen. Als $P=NP$, dan zullen de practici het vak serieus nemen. Als $P\neq NP$, dan zullen de theoretici het vak serieus nemen.

11

Met het moeilijk maken van makkelijke problemen wordt makkelijk geld verdiend. Met het makkelijk maken van moeilijke problemen is het moeilijk het hoofd boven water te houden.

12

Velen eigenen zich het recht toe anderen op fouten te betrappen. Slechts weinigen zijn in staat fouten bij zichzelf op te sporen.