Theory and Methodology

A dynamic programming algorithm for the local access telecommunication network expansion problem

Olaf E. Flippo ¹, Antoon W.J. Kolen, Arie M.C.A. Koster ², Robert L.M.J. van de Leensel *

Department of Quantitative Economics, Maastricht University, P.O. Box 616, 6200 MD Maastricht, The Netherlands

Received 5 October 1998; accepted 13 June 1999

Abstract

In this paper we consider the local access telecommunication network expansion problem, in which growing demand can be satisfied by expanding cable capacities and/or installing concentrators in the network. The problem is known to be NP-hard. We prove that the problem is weakly NP-hard, and present a pseudo-polynomial dynamic programming algorithm for the problem, with time complexity O(nB²) and storage requirements O(nB), where n refers to the size of the network, and B to an upper bound on concentrator capacity. The cost structure in the network is assumed to be decomposable, but may be non-convex, non-concave, and node and edge dependent otherwise. This allows for incorporation of many aspects occurring in practical planning problems. Computational results indicate that the algorithm is very efficient and can solve medium to large scale problems to optimality within (fractions of) seconds to minutes. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Dynamic programming; Telecommunication networks; Optimization

1. Introduction

Over the last decade major developments have occurred in the area of telecommunications. On the one hand, there has been a continuous growth in both the number of customers using telecommunication services and the user intensity with respect to these services. On the other hand, technological innovations in, for instance, transmission and switching technologies, have led to the possibility of cost reduction, as well as to improved accessibility of existing services. In addition, deregulation of the telecommunication industry has made the market more competitive, which, combined with technological advances, has resulted in a diversity of new services, such as data transmission and video applications. Due to these...
developments, enormous investment and cost saving opportunities in the design and expansion of computer communication networks and telecommunication systems have arisen. For an overview of various problems in this field we refer to [14,21].

The operations research literature reports fruitful combinatorial approaches to a wide range of network design problems. In layered networks for instance, hierarchical problems arise (see [3,2]), where nodes in higher levels must be connected by more reliable arcs or by arcs with larger capacity. Alternatively, connectivity requirements may be formulated in terms of the design of arc or node disjoint paths in the network (see for instance [16,17]). Sancho [23] considers a hierarchical network design problem in which two predetermined nodes are to be connected via a ‘backbone’ path, while the remaining nodes should be connected to this path in a tree-like structure. A dynamic programming algorithm is used to find an approximate solution for the problem.

A second class of network design problems involves multi-commodity flows, in which arcs typically have a fixed charge cost structure per commodity. Hellstrand et al. [18] discuss a formulation with a quasi-integral polytope, which allows the problem to be solved by a modified pivoting scheme. A similar problem can be found in [8,9], where a cutting-plane approach is applied to tackle the problem.

Since growing demand has led to capacity problems in existing networks, a lot of research focuses on network capacity expansion problems. Ahuja et al. [1] study the optimal expansion of transshipment networks so as to minimize the costs of network flow, subject to a constraint on the AMOUNT of expansion costs. Chang and Gavish [10] describe a multi-period expansion problem, where future demand is known for a number of periods, and the present value of future expansion costs is to be minimized (see also [19,24]).

Design and expansion problems arise frequently in local access telecommunication networks (LATNs). The LATN is a tree which connects user nodes to a switching center located in the root of the tree. Since all traffic between user nodes is routed via the switching center, demand between pairs of user nodes can be modeled as if each user node has a traffic demand that must be routed to the switching center. If the number of users in the LATN is small, a star configuration in which all users are directly connected to the central component is often employed. As the number of users increases, however, network costs can be reduced by introducing a more general tree structure, and by installing concentrators in the network. Together with the central component, these concentrators could form a backbone network (ring structure) to increase reliability, or they can also simply be connected to the central component via a direct line.

To optimize the design of these networks, a two-phase procedure is often proposed. In the first phase the user nodes are partitioned into regions, while in the second phase concentrators are placed in each of these regions to meet traffic demand between user nodes (see for instance [15,22]).

As demand increases, and the current capacity of the network no longer suffices to meet demand, the expansion of the capacity of the LATN can be achieved by upgrading the capacity of cables in the network, and/or by installing multiplexers (or concentrators) that convert signals from multiple copper cables into signals which can be carried very efficiently on relatively few fiber optic cables. Thus, various possible network settings could be exploited to meet growing demand. Since a large proportion of overall investment in telecommunication facilities is due to LATNs, efficient planning methods for the capacity expansion can yield significant cost savings. For an overview of different capacity expansion problems in LATNs we refer to [4,7,13].

In [5] the local access network expansion problem (which we will refer to as LATNEP henceforth) with a specific cost structure is introduced. They mention that the problem is NP-hard and use Lagrangian relaxation, valid inequalities and preprocessing techniques to determine (near-) optimal solutions. For the special case where existing edge capacities are equal to zero (i.e., the design problem) and which only involves one uncapacitated concentrator type with a piecewise-linear and concave cost structure, they show that
the problem is solvable in polynomial time (see also [6]). Cho and Shaw [11] studied LATNEP with a fixed charge cost structure, and solve the problem with a dynamic programming algorithm that is embedded in a column generation approach. For the design problem, their algorithm solves the problem in \(O(n^2B)\) time complexity and storage space, with \(n\) referring to the number of nodes in the tree, and \(B\) to an upper bound on concentrator capacity. For the more general expansion problem a similar approach is proposed, but the resulting algorithm is incorrect, since it may fail to find an optimal solution for certain problem instances (cf. Section 3).

In this paper, we show that LATNEP is weakly NP-hard; we present a dynamic programming algorithm for LATNEP which runs in \(O(nB^2)\) time and requires \(O(nB)\) storage space. To the authors’ knowledge, this is the first exact pseudo-polynomial time algorithm for LATNEP. Our algorithm can also handle more general cost structures for cable expansion and concentrator installation than previously considered in literature. These structures include non-convex and non-concave costs, which may be node and edge dependent. This allows us to incorporate many aspects occurring in practical planning problems, such as the availability of different electronic devices to perform multiplexing operations, the possibility to install multiple concentrators in a node, or even the demolition of concentrators, among others. The only assumption we impose is decomposability, i.e., total cable expansion (concentrator installation) costs are the sum of the individual expansion (installation) costs per edge (node). Computational experiments indicate that the proposed algorithm is very efficient; networks up to 30 nodes (as mentioned in [11]) can be solved within fractions of seconds, whereas significantly larger instances up to 1000 nodes can be solved within (fractions of) seconds to minutes, depending on network structure and concentrator capacity. Balakrishnan et al. [5] tested their method on three realistic problem instances from industry. The largest problem instance, a 41 node problem, could not be solved to optimality using their method (they report a 7% optimality gap after 15 minutes of running time). For confidentiality reasons we were not able to obtain the exact cost functions used by Balakrishnan et al. [5], but since the running time of our algorithm is basically independent of the cost structure (cf. Section 4), we could test our method on this problem instance using a variety of different cost functions. In all cases our algorithm solves the problem instance to optimality in less than a minute.

The remainder of this paper is organized as follows. In Section 3 we give a detailed problem description and a mathematical formulation. In Section 3 we embed the problem into two parameterized families of subproblems, and we derive relations between the members of these families on which the dynamic programming algorithm is based. Moreover, we justify our parameterization with an example. The algorithm itself is stated in Section 4, together with a proof of its correctness. Computational results are reported in Section 5. Final remarks and issues for future research in Section 6 conclude the paper.

2. Problem description

The local access telecommunication network is a tree which connects user nodes to the switching center located in the root of the tree. Each user node typically represents a collection of individual users connected by an underlying network. Communication between user nodes of this and other LATNs is accomplished through the switching center. Therefore, instead of using traffic demand between pairs of nodes, we can assume that each user node has a demand which must be routed to the central switching center. The demand of a user node is usually measured in the required number of circuits needed between the node and the switching center, where each circuit requires one twisted copper cable in conventional copper networks. Routing demand may be accomplished in two ways, viz., either by using dedicated cables from the user node to the switching center (via its unique path in the tree), or by routing the demand to a compression device called a concentrator, which is (to be) installed in a node of the network. A concentrator compresses all incoming low frequency signals (demand) into one outgoing high
frequency (or optical) signal, which is then routed to the switching center. It is assumed that this outgoing signal either requires negligible capacity in the network, or is routed to the switching center via a dedicated line not belonging to the network. The costs of constructing such dedicated lines are included in the installation costs of the concentrators involved. In practice, a large variety of electronic devices are available to compress signals. For the problem at hand, these different technologies can simply be treated as concentrators with different capacities and operational costs.

Due to the introduction of new services, the increased intensity in the number of customers and the continuously growing utilization of telecommunications services, the existing capacity in the LATN may no longer suffice to accommodate this increasing demand. In that case, the objective is to expand cable capacity and/or install concentrators, and possibly reroute traffic demands from user nodes, in such a way that all demand is satisfied, and the costs of the network expansion plan are minimized. Hence, the key issue in LATNEP is to find an efficient trade-off between cable expansion and concentrator installation costs.

Let $\mathcal{F} = (\mathcal{V}, \mathcal{E})$ be the tree on which LATNEP is defined, with

$$\mathcal{V} = \{0, \ldots, n\}$$

and

$$\mathcal{E} = \{1, \ldots, n\}.$$

We assume that both nodes and edges are numbered in a depth-first-search order. The predecessor of $v$ is denoted by $p_v$. Hence, edge $e \in \mathcal{E}$ equals $\{p_v, v\}$, where $v \in \mathcal{V}$ and $e$ have the same (numerical) label. For $v, w \in \mathcal{V}$, let $V(v, w)$ and $E(v, w)$ denote the nodes and edges on the path from $v$ to $w$ in $\mathcal{F}$, respectively. For every node $v \in \mathcal{V}$ a traffic demand (load) $r_v$ is given, and for every edge $e \in \mathcal{E}$ the existing capacity $b_e$ is known. We assume that both the traffic demands and the existing capacities are integral. Next, for each $v \in \mathcal{V}$, a real-valued cost function $K_v$ is given, where $K_v(k_v)$ specifies the concentrator costs that are involved when $k_v$ is the amount of demand to be processed by a concentrator in node $v$ (also referred to as the load on node $v$). Likewise, for every edge $e \in \mathcal{E}$ a real-valued cost function $L_e$ is given, where $L_e(\ell_e)$ specifies the cable costs that are involved when a demand (load) of $\ell_e$ is to be transferred over edge $e$. Due to the generality of these cost structures, a variety of problem characteristics can be taken into account as special cases. The situation where a concentrator with capacity $b_v$ is already operational in node $v$ for instance, can be accounted for by setting $K_v$ equal to the costs of installing and operating a new or supplementary concentrator if the (planned) load $k_v$ exceeds the current capacity $b_v$, and zero otherwise. Possible demolition costs of such an existing concentrator in $v$ could thereby also be included in $K_v$. Similarly, the situation where it is allowed to install multiple concentrators in a node can be handled by the model. Finally, the fixed charge cost structures that are considered in [5,11] can be accounted for. In both studies concentrators are available in different types. Every concentrator of type $t$ has a given capacity $b_v^t$, and (node dependent) fixed and variable installation costs $F_t$ and $c_v^t$, respectively. (On close inspection, the variable costs in [11] are assumed independent of the concentrator type $t$). This situation is recovered from our cost structure by setting $K_v$ equal to $\min\{F_t^v + k_v \cdot c_v^t \mid b_v^t \geq k_v\}$ for $k_v > 0$, and zero otherwise. Note that this cost structure therefore incorporates the variety of different concentrating devices which may be available in practical situations. The edge costs that are considered in these papers have a similar structure. For every edge $e \in \mathcal{E}$ an existing capacity $b_e$, and (edge dependent) fixed and variable expansion costs $F_e$ and $c_e$ are defined; cable expansion costs are then obtained by setting $L_e(\ell_e)$ equal to $F_e + (\ell_e - b_e) \cdot c_e$ for $\ell_e > b_e$, and zero otherwise. An example with the cost structure of Cho and Shaw [11] and only one concentrator type is given in Fig. 1.

The main issue in LATNEP is to decide for each node $v \in \mathcal{V}$ whether to route its demand to the switching center (which we refer to as the concentrator in the root node) via its unique path in the tree, or to route it to a node $w \in \mathcal{V} \setminus \{0\}$ in which a concentrator must then be installed to transmit all incoming load to the switching center.
of $\mathcal{T}$ via a dedicated line. If the load of node $v$ is routed to a concentrator in node $w$, we say that $v$ ‘homes on’ $w$ (cf. [5]). Note that we do allow backfeed, i.e., a node can also home on nodes other than those on the unique path to the root of the tree.

Apart from routing restrictions due to technical constraints, network planners often impose extra restrictions on the layout of telecommunication networks. Some of these restrictions are considered to be economically sensible, while other restrictions are considered practical for operational convenience regarding maintenance and repair. For LATNEP, the following restrictions should be incorporated (these restrictions are the same as proposed by Balakrishnan et al. [5], Cho and Shaw [11], Shulman and Vachani [24] and Jack et al. [19]):

1. **Single level concentration**: Demand is concentrated at most once before reaching the switching center in the root of the tree.

2. **Non-bifurcated routing**: For each user node its entire demand is processed by a single concentrator (possibly at the root).

3. **Contiguity condition**: If a node $v$ homes on a concentrator in node $w$, then all nodes on the path from $v$ to $w$ home on $w$.

Condition 1 reflects guidelines of network planners who, given the current relative costs of cable expansion and concentrator installation, consider multiple levels of concentration to be uneconomical. The compressed (high frequency) signal is assumed to be routed from the concentrator to the switching center using a dedicated line not belonging to the network. Conditions 2 and 3 are enforced to ensure operational convenience of maintenance and repair (for a detailed description of these conditions, we refer to [5]). Note that due to these routing restrictions, once it is known for each node $v$ on which node $w$ it homes, the complete configuration of the network is known. Therefore, we define

$$x_{uw} = \begin{cases} 1 & \text{if node } u \text{ homes on node } w \\ 0 & \text{otherwise} \end{cases} \quad (v, w \in \mathcal{T}),$$

$$k_u = \text{the load to be processed by a concentrator in node } u \quad (v \in \mathcal{T}),$$

$$\ell_e = \text{the load to be transferred over edge } e \quad (e \in \mathcal{E}).$$

Then LATNEP reads:

$$\min \sum_{w \in \mathcal{T}} K_w(k_w) + \sum_{e \in \mathcal{E}} L_e(\ell_e) \quad (1)$$

s.t.

$$x_{uv} = 1, \quad \forall u \in \mathcal{T}, (2)$$

$$\sum_{w \in \mathcal{T}} x_{uw} = 1 \quad \forall u \in \mathcal{T},$$

$$x_{u'w} \geq x_{uw} \quad \forall u, u', w \in \mathcal{T}: u' \in V(u, w),$$

$$k_w = \sum_{u \in \mathcal{T}} r_u \cdot x_{uw} \quad \forall w \in \mathcal{T},$$

$$\ell_e = \sum_{u, w \in \mathcal{T} \setminus \mathcal{E}(u, w)} r_u \cdot x_{uw} \quad \forall e \in \mathcal{E},$$

$$k_w \leq B \quad \forall w \in \mathcal{T},$$

$$x_{uw} \in \{0, 1\}, \quad k_w \geq 0, \quad \ell_e \geq 0 \quad \forall u, w \in \mathcal{T}, \forall e \in \mathcal{E}. \quad (8)$$

The objective function in (1) defines the total costs that follow from the network expansion program $(x, k, \ell)$. As can be seen from its formulation, it propagates the decomposability assumption on costs. As for concentrator costs in node $w$, it follows from the load $k_w$ whether or not a concentrator should be installed in node $w$, and the costs $K_w(k_w)$ can be defined accordingly to describe this situation correctly. Furthermore, existing edge capacities can be incorporated as indicated before. Constraint (2) states that a concentrator (the switching center) is installed in the root node. Constraint (3) implies that every node homes on exactly one node (and thereby ensures the non-bifurcated routing condition),
whereas (4) enforces the contiguity condition. Note that (4) contains a lot of redundancy. However, since the model is only used to communicate the problem and prove the validity of our dynamic programming approach, efficiency in the number of constraints is not an issue here. Constraints (5) and (6) merely define the resulting loads on the nodes and edges, respectively. Without loss of generality we assume in (7) that for any given \( w \in \mathcal{W} \), a uniform bound \( B \) exists that restricts the sum of the loads of all nodes homing on \( w \) to \( B \). In practical situations this upper bound is given by the maximum of the capacities of the available concentrator types, or in the case of an uncapacitated concentrator by the sum of all the loads in the tree \( \mathcal{T} \). The integrality and non-negativity constraints in (8) complete the formulation. Although Balakrishnan et al. [5] and Cho and Shaw [11] use different IP formulations (they include variables to represent the cable expansion decision as well as the installation of concentrators), their papers describe the same problem. Finally, we denote the set of feasible solutions by \( \mathcal{F} \), hence

\[
\mathcal{F} = \{(x, k, \ell) \mid (x, k, \ell) \text{ satisfies (2)--(8)}\}.
\]

3. Parametrizations for LATNEP

In this section we introduce two parameterized families of subproblems, which will be defined on subtrees of \( \mathcal{T} \). The main idea is that each of the subtrees we consider (unless it consists of a single node), is decomposed into two smaller subtrees. Optimal solutions for subproblems defined on the latter two subtrees are then used to obtain optimal solutions for subproblems defined on the former.

3.1. Defining subtrees \( T[v,i] \) of tree \( \mathcal{T} \)

The subtrees we employ were introduced by Johnson and Niemi [20] to efficiently solve tree knapsack problems and tree partitioning problems. Let \( d_i \) be the number of children (successors) of node \( v \) in \( \mathcal{T} \), and \( D_e = \{s^i_1, \ldots, s^i_{d_i}\} \) the set of its children, with \( s^i_k \) the \( k \)th child of \( v \). For \( v \in \mathcal{V} \) and \( 0 \leq i \leq d_i \), we define the subtree \( T[v,i] \), which is induced by node \( v \), its first \( i \) children \( \{s^i_1, \ldots, s^i_i\} \) and all successors of these children (see [20]). For example, in the tree of Fig. 1, \( T[2,1] \) is given by the subtree defined on the nodes 2, 3, 4, 5, whereas \( T[2,2] \) is given by the subtree defined on the nodes 2, 3, 4, 5, 6. Note that \( T[0,d_i] \) is the complete tree \( \mathcal{T} \), and \( T[v,0] \) is the subtree of \( \mathcal{T} \) consisting only of the node \( v \). Also observe, that for \( v \in \mathcal{V} \) and \( 1 \leq i \leq d_i \), the tree \( T[v,i] \) consists of the subtrees \( T[v,i-1] \) and \( T[s^i_i,d_i] \), together with the edge \( \{v,s^i_i\} \). Finally, we let \( V[v,i] \) be the node set of \( T[v,i] \).

3.2. Defining subproblems on subtrees

Given an arbitrary subtree \( T = (V,E) \) of \( \mathcal{T} \) and an arbitrary solution \( (x,k,\ell) \in \mathcal{T} \), we define the costs of subtree \( T \) for the solution \( (x,k,\ell) \) by

\[
C(x,k,\ell \mid T) = \sum_{w \in \mathcal{W}} K_w(k_w) + \sum_{e \in E} L_e(\ell_e).
\]

Note that for \( w \in V \), the variable \( k_w \) also contains the load from nodes that do not belong to \( T \), but that do home on \( w \). Similarly, \( \ell_e \) may contain load from outside (inside) \( T \) that is transferred over \( e \) to a concentrator inside (outside) \( T \). In the first family of subproblems we restrict ourselves to the case in which the root of the subtree homes on a node inside the subtree: for \( (v,i) \) with \( v \in \mathcal{V} \) and \( 0 \leq i \leq d_i \) let

\[
g(v,i,s) = \min \ C(x,k,\ell \mid T[v,i])
\]

s.t.

\[
\begin{align*}
x_{vw} &= 0 & \forall w & \notin V[v,i], \\
\sum_{w \in V[v,i]} \sum_{e \in E} r_{uw} \cdot x_{uw} &= s, \\
(x,k,\ell) & \in \mathcal{T}.
\end{align*}
\]

Hence, \( g(v,i,s) \) represents the minimal costs of subtree \( T[v,i] \) among all solutions \( (x,k,\ell) \in \mathcal{T} \) for which \( v \) homes within \( T[v,i] \), and the total demand from nodes not in \( T[v,i] \) homing within \( T[v,i] \) equals \( s \). Note that by contiguity, this load \( s \) must home on the same node as node \( v \) does (say node \( \bar{w} \)), which explains the upper bound on \( s \). More-
over, in order to determine the concentrator costs in node \( \tilde{v} \), the complete load which has to be processed by this concentrator has to be known. Since part of this complete load may be due to demand from nodes outside \( T[v, i] \), the parameter \( s \) is incorporated in the parameterization. Finally, note that \( g(0, d_0, 0) \) is equivalent to LATNEP, hence, this is the problem we ultimately want to solve.

In the second family of subproblems we restrict ourselves to the case in which the root of the subtree homes on a node outside the subtree: for \( (v, i) \) with \( v \in V, \ 0 \leq i \leq d_v \) and \( r_v \leq r \leq \min(B, \sum_{w \in V[v,i]} r_w) \) we define

\[
h(v, i, r) = \min C(x, k, \ell \mid T[v, i])
\]

s.t.

\[
x_{vw} = 0 \quad \forall w \in V[v, i],
\]

\[
\sum_{w \in V[v,i], w \notin V[v,i]} r_w \cdot x_{uw} = r,
\]

\[
(x, k, \ell) \in \mathcal{F}.
\]

So, \( h(v, i, r) \) represents the minimal costs of subtree \( T[v, i] \) among all solutions \( (x, k, \ell) \in \mathcal{F} \) for which node \( v \) does not home within \( T[v, i] \) and the total demand of nodes in \( T[v, i] \) homing outside \( T[v, i] \) equals \( r \). Again, note that by contiguity this load \( r \) must home on the same node as node \( v \).

It is important to note that the formulations (10)–(13) and (14)–(17), for \( g(v, i, s) \) and \( h(v, i, r) \), respectively, may not have feasible solutions, since constraints (12) and (16) may be impossible to satisfy. For example, if the demand \( r_v \) is even for all user nodes \( v \), then these constraints are impossible to satisfy if either the parameter \( s \) or \( r \) is odd. In this case the corresponding \( g \) or \( h \) value is not real but infinity. For simplicity, in the sequel we will refer to such a situation by saying that the ‘problem’ \( g(v, i, s) \) or \( h(v, i, r) \) is infeasible. Similarly, if the problem is feasible, we will say that \( g(v, i, s) \) or \( h(v, i, r) \) has a feasible solution. We will return to this issue on several occasions in the sequel; firstly in Section 3.3, when the recursive relations on which the dynamic programming algorithm is based, are discussed, and secondly in Section 4, when the correctness of the algorithm is considered.

### 3.3. Relations between family members

First, we discuss the starting point of the dynamic programming algorithm, which is given by subtrees \( T[v, 0] \) consisting only of the node \( v \) itself. Both for the \( g \)- and \( h \)-coefficients, the minimal costs can be determined easily in this case, as is described in the following propositions.

**Proposition 3.1.** Consider \( (v, i) \) with \( v \in V \) and \( i = 0 \). If \( g(v, 0, s) < \infty \) then

\[
g(v, 0, s) = K_v(r_v + s). \tag{18}
\]

**Proof.** Since \( V[v, 0] = \{v\} \), it follows from (11), (3) and (8) that \( x_{vw} = 1 \). Furthermore,

\[
k_v = \sum_{w \in V} r_u \cdot x_{uw} = \sum_{w \in V[v, 0]} r_u \cdot x_{uw} + \sum_{w \notin V[v, 0]} r_u \cdot x_{uw} \tag{19}
\]

\[
= r_v \cdot x_{vv} + \sum_{w \notin V[v, 0]} r_u \cdot x_{uw} = r_v + s.
\]

The objective function in (10) thus amounts to

\[
C(x, k, \ell \mid T[v, 0]) = K_v(r + s). \quad \square
\]

Note that the condition \( g(v, 0, s) < \infty \) is needed in Proposition 3.1, since a feasible solution for \( g(v, 0, s) \) need not exist.

**Proposition 3.2.** Consider \( (v, i) \) with \( v \in V \) and \( i = 0 \). If \( h(v, 0, r) < \infty \) then \( v \neq 0 \), \( r = r_v \),

\[
r \leq B - \min_{u \neq v} \{r_u\}
\]

and

\[
h(v, 0, r) = K_v(0). \tag{19}
\]

**Proof.** From (2) and (15) it follows directly that \( v \neq 0 \). Moreover, \( V[v, 0] = \{v\} \) and (16) imply that \( r = r_v \), and since the load \( r \) must home on a node \( w \) outside \( T[v, i] \), by contiguity the first node on the path from \( v \) to \( w \) homes on \( w \). Hence,

\[
r \leq B - \min_{u \neq v} \{r_u\}.
\]
Finally, the objective function in (14) amounts to
\[ C(x, k, \ell | T[v, i]) = K_e(k_e)(0). \]

Our dynamic programming algorithm operates in a bottom-to-top kind of fashion as follows. Suppose that in the example of Fig. 1 subtree \( T[2, 2] \) is under consideration for the calculation of \( g(2, 2, s) \). Since \( T[2, 2] \) is composed of the two subtrees \( T[2, 1] \) and \( T[6, 0] \) (along with edge 6), the general idea is to obtain an optimal solution to the former by combining optimal solutions of the latter two. Since in \( g(2, 2, s) \) node 2 must homed within \( T[2, 2] \), two situations may arise. On the one hand, it may be optimal to combine an optimal solution of \( T[6, 0] \) with node 6 homing on some node in \( T[6, 0] \) (necessarily node 6 in this example), with an optimal solution of \( T[2, 1] \) with node 2 homing on some node in \( T[2, 1] \). In this case the former of the two optimal solutions is an optimal solution for \( g(6, 0, 0) \) and the latter is an optimal solution for \( g(2, 1, s) \). On the other hand it may be optimal to combine optimal solutions of \( T[6, 0] \) and \( T[2, 1] \) with nodes 2 and 6 both homing on the same node \( w \). In case \( w \in V[2, 1] \), the load from \( T[6, 0] \) is transferred to \( T[2, 1] \) via edge 6. Since the resulting costs depend on the load that is transferred over edge 6, it is necessary to know how much load is actually involved. Let \( x \) denote the load over edge 6, then the solution for \( g(2, 2, s) \) can be obtained by combining solutions \( h(6, 0, x) \) and \( g(2, 1, s + x) \). Similarly, if \( w \in V[6, 0] \) then the load is transferred from \( T[2, 1] \) to \( T[6, 0] \) via edge 6, and the solution for \( g(2, 2, s) \) can be obtained by combining solutions \( h(6, 0, s + x) \) and \( h(2, 1, x) \). For the calculation of \( h(2, 2, r) \) several cases can be distinguished in a similar manner. These ideas are now formalized for the general case in which we combine solutions of the subtrees \( T[v, i - 1] \) and \( T[x_v, d_v] \) to obtain a solution for the subtree \( T[v, i] \) in Proposition 3.3 and 3.4 for \( g(v, i, s) \) and \( h(v, i, r) \), respectively. For a rigorous analysis and strict mathematical proof of the propositions we refer to [12].

**Proposition 3.3.** Consider \((v, i)\) with \(v \in \mathcal{V}\) and \(1 \leq i \leq d_v\). Define
\[
A_v = g(s_v^i, d_v^i, s + x) + h(v, i - 1, x) + L_v^i(s + x)
\]
\[ (r_v \leq x \leq B - s - r_v^i) , \]
\[ B_v = h(s_v^i, d_v^i, x) + g(v, i - 1, s + x) + L_v^i(x)
\]
\[ (r_v^i \leq x \leq B - s - r_v^i), \]
\[ C = g(s_v^i, d_v^i, 0) + g(v, i - 1, s) + L_v^i(0), \]
\[ D = \min \left[ \min_{r_v \leq x \leq B - s - r_v^i} \{A_v\}, \min_{r_v^i \leq x \leq B - s - r_v} \{B_v\}, C \right]. \]

Then \(g(v, i, s) \geq D\). Moreover, if \(g(v, i, s) < \infty\) then \(g(v, i, s) = D\).

**Proposition 3.4.** Consider \((v, i)\) with \(v \in \mathcal{V}\) and \(1 \leq i \leq d_v\). Define
\[
E_v = h(s_v^i, d_v^i, x) + g(v, i - 1, r - x) + L_v^i(x)
\]
\[ (r_v^i \leq x \leq r_v - r_v^i), \]
\[ F = g(s_v^i, d_v^i, 0) + g(v, i - 1, r) + L_v^i(0), \]
\[ G = \min \left[ \min_{r_v^i \leq x \leq B - s - r_v} \{E_v\}, F \right]. \]

Then \(h(v, i, r) \geq G\). Moreover, if \(h(v, i, r) < \infty\) then \(h(v, i, r) = G\).

Note that the partition of the space of feasible solutions might lead to the idea that the ‘greater than or equal’-sign in \(g(v, i, s) \geq D\), as formulated in Proposition 3.3, can be replaced by an equality. However, this is not the case. To illustrate this, suppose that all demand values from nodes outside of \(T[v, i]\) are even, and demand values for nodes within \(T[v, i]\) have arbitrary values. Next, assume that the coefficient \(g(v, i, s)\) is determined with \(s\) odd, and the minimum for \(D\) is achieved by \(A_v\) for some odd number \(x\). To see that this may actually occur, first note that an even load \(s + x\) can be achieved by nodes outside the tree \(T[x_v, d_v]\), since demand figures in \(T[v, i - 1]\) have arbitrary values, which will give a feasible solution for \(g(s_v^i, d_v^i, s + x)\). Secondly, an odd load from within the tree \(T[v, i - 1]\) can be achieved (again because of the arbitrary demand values within \(T[v, i - 1]\)) which then yields a feasible solution for \(h(v, i, x)\). However, since all demand figures for nodes not in \(T[v, i]\) are even, there will not exist a feasible
solution for \(g(v, i, s)\) with \(s\) odd, which implies that 
\[ g(v, i, s) = \infty. \]
Hence, equality does not necessarily
hold in Proposition 3.3. A similar conclusion applies to
'standard' dynamic programming algorithms, the
correctness of our algorithm based on the above relations is not immediately clear. In Section 4 we
will state a mathematical proof of the algorithm’s
validity.

Moreover, note that in the absence of the pa-
rameters \(s\) and \(r\), an algorithm will result which
may fail to calculate the optimal solution. Con-
sider, for example, the problem in Fig. 1. Observe
that in the case node 1 does not home on node 4,
the costs will be infinite. By contiguity the nodes 2,
3 and 4 must then also home on 4. In that case the
capacity of edge 3 must be expanded and the load
on edge 4 is at least 16, which leaves a slack of 5 on
that edge. The latter observation implies that it is
impossible to home both 5 and 6 on 4 without
incurring costs of infinity. So, there are three
possible solutions for the overall problem with cost
less than infinity, viz.,

(i) nodes 1, 2, 3, 4 and 6 home on 4, nodes 0 and
5 home on themselves; costs 60;
(ii) nodes 1, 2, 3, 4 and 5 home on 4, nodes 0 and
6 home on themselves; costs 65;
(iii) nodes 1, 2, 3 and 4 home on 4, nodes 0, 5
and 6 home on themselves; costs 70.

If we consider the solutions of the subtree
\(T[2, 2]\) with 2 homing inside the tree, we have the
choice between

(iv) nodes 2, 3, 4 and 6 home on 4, node 5 home
on itself; partial costs 48;
(v) nodes 2, 3, 4 and 5 home on 4, node 6 home
on itself; partial costs 47;
(vi) nodes 2, 3 and 4 home on 4, nodes 5 and 6
home on themselves; partial costs 52.

Note that in both solutions (iv) and (v) the total
load of nodes in \(T[2, 2]\) that home on node 4 equals
19. What makes (iv) more expensive, is the fact
that the fixed costs of expanding edge 3 are in-
curred, whereas in the latter solution they are not.
However, this is exactly the reason why the ‘cur-
rently non-optimal’ solution (v) is better suited to
receive the additional load from node 1 at the next
stage of the algorithm. Hence, in the absence of a
parameter \(s\) that represents the load that may enter
the current subtree during a later stage of the al-
gorithm, solutions as in (iv) may be eliminated
from further consideration. The example shows
that, unfortunately, the overall optimal solution
may then be eliminated as well.

4. An \(O(nB^2)\) dynamic programming algorithm for
LATNEP

The relationships between the subproblems
derived in the preceding section give rise to the
following dynamic programming algorithm. Re-
call that \(g(v, i, s)\) is only defined for \((v, i)\) with
\(v \in \mathcal{V}^*, 0 \leq i \leq d_v\) and \(0 \leq s \leq B - r_v\), whereas
\(h(v, i, r)\) is defined on the same \((v, i)\) pairs (ex-
cluding \(v = 0\), since a concentrator is always in-
stalled in the root) with \(r_v \leq r \leq B\).

**DYNAMIC PROGRAMMING ALGORITHM FOR LATNEP**

\[
\text{forall } (v, i, s) \text{ with } v \in \mathcal{V}^*, 0 \leq i \leq d_v \text{ and } 0 \leq s \leq B - r_v \text{ do } \\
g(v, i, s) = \infty; \quad \text{/* initialization } \text{g} */
\]

\[
\text{forall } (v, i, r) \text{ with } v \in \mathcal{V}^*, 0 \leq i \leq d_v \text{ and } r_v \leq r \leq B \text{ do } \\
h(v, i, r) = \infty; \quad \text{/* initialization } \text{h} */
\]

\[
\text{forall } v = n \text{ downto } 0 \text{ do begin } \\
\text{forall } s \text{ with } 0 \leq s \leq B - r_v \text{ do } \\
g(v, 0, s) = K_v(r_v + s);
\]

\[
\text{if } (v \neq 0) \text{ then } \\
h(v, 0, r_v) = K_v(0);
\]

\[
\text{forall } i = 1 \text{ to } d_v \text{ do begin } \\
\text{forall } s \text{ with } 0 \leq s \leq B - r_v \text{ do } \\
g(v, i, s) = D \text{ with } D \text{ defined as in } (20)\text{–}(23);
\]

\[
\text{forall } r \text{ with } r_v \leq r \leq B \text{ do } \\
h(v, i, r) = G \text{ with } G \text{ defined as in } (24)\text{–}(26);
\]

\[
\text{end};
\]

\[
\text{end};
\]

optimal solution: \(g(0, d_0, 0)\)

Unfortunately, as already indicated in the pre-
vious section, the correctness of the algorithm does
not immediately follow from the results in Section
3.3, since Propositions 3.3 and 3.4 do not ex-
clude the possibility that \(D < g(v, i, s) = \infty \) or
$G < h(v, i, r) = \infty$. Indeed, this situation may occur, since we ignore constraints (12) and (16) during computations. In Theorem 4.1 we give a mathematical proof of the correctness of the algorithm. In the forthcoming analysis we will distinguish between the values of $g(v, i, s)$ and $h(v, i, r)$ as defined in Section 3.3 on the one hand, and the ones that are computed by the aforementioned algorithm on the other hand, by temporarily providing the latter with a superscript $c$ (of 'computed'). Roughly speaking, we prove that the computed values $g^c(v, i, s)$ and $h^c(v, i, r)$ may differ from the true values $g(v, i, s)$ and $h(v, i, r)$ only if these problems are infeasible, which is sufficient to prove the main result of this paper (summarized in Theorem 4.2).

**Theorem 4.1.** Consider $(v, i)$ with $v \in T'$ and $0 \leq i \leq d_v$. Let $r$ and $s$ be such that $r_v \leq r \leq B$ and $0 \leq s \leq B - r_v$. Then the following statements hold:

(i) If $g(v, i, s) < \infty$ then $g^c(v, i, s) = g(v, i, s)$;

(ii) If $r = B - \min_{u \in \{i, j \mid |u| \in \varepsilon\}} \{r_u\}$ then $h^c(v, i, r) = h(v, i, r)$.

**Proof.** First consider $i = 0$. If $g(v, i, s) < \infty$ then $g^c(v, i, s) = g(v, i, s)$ follows from (18) and the definition of $g^c(v, i, s)$ in the algorithm. If $r = B - \min_{u \in \{i, j \mid |u| \in \varepsilon\}} \{r_u\}$ then Proposition 3.2 states that $h(v, i, r) = K_v(0)$ if $r = r_v$, $v \neq 0$, and $h(v, i, r) = \infty$ if $r \neq r_v$ or $v = 0$. Hence, $h^c(v, i, r) = h(v, i, r)$ follows from (19) and the definition of $h^c(v, i, r)$ in the algorithm.

To complete the proof we use induction on the pairs $(v, i)$ with $i > 0$ in the order as described by the algorithm. If $g(v, i, s) < \infty$ then by Proposition 3.3 we have $g(v, i, s) = D$. If the minimum in (23) is attained for $A_\tilde{s}$ for some $\tilde{s}$ with $r_v \leq \tilde{s} < B - r_v$, then

$$g(v, i, s) = g^c(s', d', s + \tilde{s}) + h(v, i - 1, \tilde{s}) + L(d')(s + \tilde{s})$$

where the second equality follows from the induction hypothesis, since $g(s', d', s + \tilde{s}) < \infty$ and $\tilde{s} \leq B - r_v \leq B - r_v \leq B - \min_{u \in \{i, j \mid |u| \in \varepsilon\}} \{r_u\}$.

If the minimum in (23) is attained for $B_\tilde{s}$ for some $\tilde{s}$ or for $C$, then $g(v, i, s) \geq g^c(v, i, s)$ follows similarly. Next we will prove the reverse inequality.

The above mentioned yields $g^c(v, i, s) \leq g(v, i, s)$ if and only if $g^c(v, i, s) < \infty$. If the minimum for $g^c(v, i, s)$ is attained by (20), then

$$g^c(v, i, s) = g^c(s', d', s + \tilde{s}) + h^c(v, i - 1, \tilde{s}) + L(d')(s + \tilde{s})$$

where the second equality is explained as follows. First, since the minimum is attained by (20), $g^c(s', d', s + \tilde{s})$ is actually computed. From the algorithm it then follows that $s + \tilde{s} \leq B - r_v$ which implies that $\tilde{s} \leq B - s - r_v \leq B - \min_{u \in \{i, j \mid |u| \in \varepsilon\}} \{r_u\}$.

By the induction hypothesis we can therefore conclude that $h^c(v, i - 1, \tilde{s}) = h(v, i - 1, \tilde{s})$. Secondly, since $g(v, i, s) < \infty$, $h(v, i - 1, \tilde{s}) < \infty$, $s \leq B - \tilde{s} - r_v$ and $\tilde{s} \geq r_v$, we can construct a feasible solution for $g^c(s', d', s + \tilde{s})$ using feasible solutions of $g(v, i, s)$ and $h(v, i - 1, \tilde{s})$ (see [12] for the strict mathematical construction of this solution). From the induction hypothesis it follows that $g^c(s', d', s + \tilde{s}) = g(s', d', s + \tilde{s})$, which justifies the second equality.

If the minimum for $g^c(v, i, s)$ is attained by (21) for some $\tilde{s}$ or by (22), then $g^c(v, i, s) \geq g(v, i, s)$ follows similarly. As a result, $g(v, i, s) < \infty$ implies $g^c(v, i, s) = g(v, i, s)$. For $h(v, i, r)$, the result follows in a similar way. \[\square\]

**Theorem 4.2.** Suppose $K_v(k_v)$ can be computed in $O(m)$ time for every $k_v \in [r_v, B] \cap \mathbb{N}$ and $v \in T$, and $L_v(k_v)$ can be computed in $O(p)$ time for every $k_v \in [0, B] \cap \mathbb{N}$ and $v \in \varepsilon$, where $m$ and $p$ are parameters depending on problem size. Furthermore, let each of these computations require $O(nB)$ storage space. Then the aforementioned dynamic programming algorithm finds an optimal solution in
O(n(m + p)B + nB^2) time and O(nB) storage space. Under mild conditions on the cost structure (which are satisfied in the real-life applications of Balakrishnan et al. [5] and Cho and Shaw [11]), it follows that O(m) = O(B) and O(p) = O(1), implying an overall time complexity of O(nB^2).

**Proof.** Correctness follows directly from $g^*(0, d_0, 0) = g(0, d_0, 0)$ (cf. Theorem 4.1). As for time and space complexity, all coefficients $K_e(k_e)$ can be calculated in $O(nmB)$ and all coefficients $L_i(\ell_i)$ in $O(npB)$. Storage requirements for these coefficients is $O(nB)$. Since the number of $(v, i)$-pairs we consider is $O(n)$, it follows that all remaining computations can be done in $O(nB^2)$. Obviously, the storage requirement for all $g$ and $h$ coefficients equals $O(nB)$.

In practical situations it is reasonable to assume that $O(m) = O(B)$. For instance, in the cost structure described by Cho and Shaw [11] (cf. Section 2), the variable concentrator costs $\tilde{c}_v$ do not depend on the concentrator type $t$. So, if two concentrators $t_1$ and $t_2$ have the same capacity $\tilde{b}^{t_1} = \tilde{b}^{t_2}$, and if $\tilde{F}_v^{t_1} \leq \tilde{F}_v^{t_2}$, then $t_1$ will never be more expensive to use than $t_2$. In other words, for every node $v \in V$ there will be at most one non-dominated concentrator per load figure $k_e$, implying that $\min\{F_v^e + \tilde{c}_v k_e \mid \tilde{b}^{t_e} \geq k_e\}$ can be computed in $O(m) = O(B)$ time. Similarly, in [5], the concentrator cost structure is represented by a piecewise-linear, concave function, with breakpoints occurring only at integer-valued arguments. Since such a function is described as the point-wise minimum of at most $B$ affine functions, it follows again that $O(m) = O(B)$. □

Now we have established the correctness of the procedure, we will drop the superscript $c$ henceforth. Note that the algorithm in its current form only determines the optimal solution value, rather than an optimal solution. However, from Propositions 3.1–3.4 it follows that an optimal solution can be recovered in the traditional way by keeping track of the ‘argmin’ (instead of just the ‘min’) in the evaluations of $g$ and $h$ coefficients, and by constructing the solution afterwards using backward recursion.

**5. Computational results**

To assess the practical feasibility of the dynamic programming algorithm, we have implemented the algorithm in the programming language C++ on a DEC 2100 A500MP workstation with 128 Mb internal memory. We tested the algorithm both on generated instances resembling Cho and Shaw’s [11] and on the real-life 41 node instance of Balakrishnan et al. [5].

**5.1. Generated problem instances**

All generated problem instances have the same cost structure as the one proposed by Cho and Shaw [11] (cf. Section 2). The first set of instances we consider has been made publicly available by Cho and Shaw. The number of nodes in the tree varies from 5 to 30, whereas the maximum concentrator capacity is in the range of 41–951. The results in Table 1 indicate that our algorithm is competitive to Cho and Shaw’s algorithm; it is 43 times as fast on average over all 11 problems, and even up to 76 times as fast on average over the 5 largest problems. (Note that this may partly be due to differences in hardware; Cho and Shaw used a SUN SPARC 1000 workstation.)

We also tested 10 larger instances of this type using the same generator as Cho and Shaw (available on Shaw’s Web site), with the number of nodes varying from 50 to 200, and with the maximum concentrator capacity varying from 100 to 1500. Our results for these instances are listed in Table 2, which indicate that much larger instances can still be solved efficiently: instances with small concentrator capacities are solved within a second, whereas the running time for larger concentrator capacities is within minutes.

All of the above instances consisted of trees with an unbalanced structure as is illustrated in Fig. 2. Due to this structure, the number of coefficients $r$ for which $h(v, i, r)$ is feasible for the $(v, i)$-pairs with $v \in \{0, 16, 31, 35, 45\}$ is very large, which makes this type of instances relatively hard to solve. Therefore, we slightly modified Cho and Shaw’s problem generator so as to obtain more balanced trees. We generated trees consisting of 25
up to 1000 nodes, and concentrator capacities ranging from 3000 to over 10 000. For each of the problem sizes, we generated five instances with 3 concentrator types. The best, average and worst CPU times are reported in Table 3. The results indicate that small problem instances can be solved in a second, whereas the larger instances can be solved in a minute. An implementation of the algorithm as well as the modified generator with all of the aforementioned instances are publicly available on World Wide Web or by e-mail to one of the authors.

5.2. Real-life problem instances

Balakrishnan et al. [5] propose a solution method which incorporates valid inequalities in a dynamic program to solve the uncapacitated version of the problem (i.e., the design problem). For three realistic networks they embed their dynamic program in a Lagrangian relaxation scheme to obtain solutions for the capacitated version (i.e., LATNEP) within 1.2–7.0% of optimality and 15 min of running time. Unfortunately, these

![Fig. 2. The tree of problem LATNEP50a.](image)

Table 1

<table>
<thead>
<tr>
<th>Problem</th>
<th>n</th>
<th>m</th>
<th>B</th>
<th>Value</th>
<th>CPU seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp5a</td>
<td>5</td>
<td>1</td>
<td>41</td>
<td>559</td>
<td>0.017</td>
</tr>
<tr>
<td>exp9a</td>
<td>9</td>
<td>1</td>
<td>51</td>
<td>1305</td>
<td>0.020</td>
</tr>
<tr>
<td>exp9b</td>
<td>9</td>
<td>1</td>
<td>70</td>
<td>559</td>
<td>0.020</td>
</tr>
<tr>
<td>exp9c</td>
<td>9</td>
<td>1</td>
<td>340</td>
<td>25847</td>
<td>0.023</td>
</tr>
<tr>
<td>exp15a</td>
<td>15</td>
<td>3</td>
<td>451</td>
<td>55566</td>
<td>0.028</td>
</tr>
<tr>
<td>exp15b</td>
<td>15</td>
<td>3</td>
<td>474</td>
<td>65693</td>
<td>0.039</td>
</tr>
<tr>
<td>exp20a</td>
<td>20</td>
<td>1</td>
<td>140</td>
<td>23382</td>
<td>0.029</td>
</tr>
<tr>
<td>exp20b</td>
<td>20</td>
<td>3</td>
<td>443</td>
<td>62859</td>
<td>0.038</td>
</tr>
<tr>
<td>exp20c</td>
<td>20</td>
<td>3</td>
<td>951</td>
<td>162936</td>
<td>0.059</td>
</tr>
<tr>
<td>exp30a</td>
<td>30</td>
<td>1</td>
<td>340</td>
<td>93238</td>
<td>0.044</td>
</tr>
<tr>
<td>exp30b</td>
<td>30</td>
<td>3</td>
<td>451</td>
<td>86717</td>
<td>0.077</td>
</tr>
</tbody>
</table>

Table 2

<table>
<thead>
<tr>
<th>Problem</th>
<th>n</th>
<th>m</th>
<th>B</th>
<th>Value</th>
<th>CPU seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>LATNEP50a</td>
<td>50</td>
<td>3</td>
<td>185</td>
<td>247768</td>
<td>0.073</td>
</tr>
<tr>
<td>LATNEP50b</td>
<td>50</td>
<td>3</td>
<td>392</td>
<td>136251</td>
<td>0.312</td>
</tr>
<tr>
<td>LATNEP50c</td>
<td>50</td>
<td>3</td>
<td>928</td>
<td>127577</td>
<td>2.106</td>
</tr>
<tr>
<td>LATNEP100a</td>
<td>100</td>
<td>3</td>
<td>190</td>
<td>632303</td>
<td>0.169</td>
</tr>
<tr>
<td>LATNEP100b</td>
<td>100</td>
<td>3</td>
<td>280</td>
<td>443484</td>
<td>0.465</td>
</tr>
<tr>
<td>LATNEP100c</td>
<td>100</td>
<td>3</td>
<td>772</td>
<td>342672</td>
<td>8.304</td>
</tr>
<tr>
<td>LATNEP200a</td>
<td>200</td>
<td>3</td>
<td>137</td>
<td>1453570</td>
<td>0.243</td>
</tr>
<tr>
<td>LATNEP200b</td>
<td>200</td>
<td>3</td>
<td>192</td>
<td>1417313</td>
<td>0.490</td>
</tr>
<tr>
<td>LATNEP200c</td>
<td>200</td>
<td>3</td>
<td>680</td>
<td>795365</td>
<td>14.999</td>
</tr>
<tr>
<td>LATNEP200d</td>
<td>200</td>
<td>3</td>
<td>1424</td>
<td>574739</td>
<td>84.442</td>
</tr>
</tbody>
</table>
instances are not publicly available. However, in their article all demands and existing cable capacities for the largest (41) node instance are given. Since our algorithm is basically independent of the cost structure (assuming that the concentrator and cable cost can be calculated in $O^*(B)$ and $O^*(1)$ time), we tested our algorithm on the 41 node instance with a Cho and Shaw cost structure. In Table 4 the computation times for several values of $B$ are given; the largest $B$ is hereby set equal to the sum of all demand. We also tested this instance on a Pentium 166 MHz personal computer with 16 Mb internal memory, on which we needed 105 seconds to compute the optimal solution (with $B = 43, 212$).

| $n$ | min $B$ | max $B$ | CPU seconds
|-----|---------|---------|--------------
| 25  | 4408    | 4869    | 0.315        |
| 50  | 4169    | 4763    | 0.707        |
| 100 | 3761    | 5360    | 1.198        |
| 200 | 7708    | 10218   | 19.889       |
| 500 | 3800    | 4762    | 6.907        |
| 1000| 2907    | 3597    | 8.479        |

Table 4

Computational results for the Balakrishnan et al. [5] 41 node instance

<table>
<thead>
<tr>
<th>$B$</th>
<th>CPU seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 000</td>
<td>2.875</td>
</tr>
<tr>
<td>20 000</td>
<td>13.655</td>
</tr>
<tr>
<td>30 000</td>
<td>33.993</td>
</tr>
<tr>
<td>43 212</td>
<td>44.600</td>
</tr>
</tbody>
</table>

6. Concluding remarks

In this paper, we have described a pseudo-polynomial time dynamic programming algorithm for the LATNEP. Our model follows a bottom-to-top approach using two parameterized families of related subproblems. We consider general cost structures which enables the model to encompass many practical situations without altering the (complexity of the) algorithm.

The computational experiments indicate that the algorithm is highly efficient. For real-life problem instances mentioned in the literature, which could not be solved to optimality, our algorithm finds an optimal solution within a minute. Significantly larger problems, with many more nodes and higher concentrator capacities, can still be solved within minutes.

For the cost structure described in Section 2, Cho and Shaw have also studied LATNEP. They also consider the special case in which all existing edge capacities are zero. This special case is referred to as the local access telecommunication network design problem (LATNDP). The typical characteristic of the LATNDP is that for each edge on which demand is routed, fixed expansion costs are necessarily incurred. By including the concentrator location $w$ into the state space, one knows that on each edge on the path from $v$ to $w$ capacity must be expanded. If an extra load from outside the tree also homes on $w$, then the extra costs on this path are determined by the variable edge costs on this path. Because of this exact (linear) relation between the costs of extra load entering the tree, this parameter can be excluded from the state space. By applying a ‘left-to-right’ instead of a ‘bottom-to-top’ approach, they are able to solve LATNDP in $O(n^2B)$. A possible combination of the ideas used in our paper with such a ‘left-to-right’ approach is a topic for further research.

References


