Single Node Service Provision with Fixed Charges* Extended Abstract

Shane Dye,[†]Leen Stougie,[‡]Asgeir Tomasgard,[§]

July 19, 1999

Abstract

The service provision problem described in this paper comes from an application of distributed processing in telecommunication networks. The objective is to maximize a service provider's profit from offering computational based services to customers. The services are built from software applications called subservices. The service provider has limited capacity of some resources and therefore must choose from a set of software subservices those he would like to offer. He maximizes profit which depends both on the number of requests met and fixed charges for installing the subservices. A fixed charge is positive when it is charged by the service provider and negative if it is an amount the service provider must pay for example to license the subservice.

Our main interest in this problem comes from it being a subproblem in a scenario decomposition of the service provision problem with stochastic demand. We assume stochasticity is represented in terms of scenarios from a discrete probability distribution. The fixed charges can be interpreted as dual prices on the linking constraints in a variable split that makes the stochastic problem separable in its demand scenarios.

The main contribution of the paper is to give a pseudo-polynomial time algorithm to solve the fixed charge problem and describe how this algorithm can be used in a fully polynomial time approximation scheme. It is also indicated how the results from this paper are used in a decomposition scheme for the stochastic service provision problem.

Keywords: distributed processing, telecommunications, service provision, dynamic programming, (stochastic) integer programming, fully polynomial approximation scheme.

1 Introduction

The service provision problem comes from an application in telecommunications. It considers how to install different *processing based services* on a set of computer nodes in a network with distributed processing capabilities. The computers typically have limited resources such as memory, processing capacity and storage capacity. All the services are built from a set of *subservices*. The subservices are software applications that run in a distributed manner in the network. These applications communicate using an underlying transportation network. The focus of this paper is on how to allocate computational resources to a set of subservices in order to maximize the profit gained through meeting customer demand for services. Because the resources are limited, it may be necessary to reject some customers.

The shift of focus from transportation to computational resources in the telecom industry follows as a consequence of three factors: technological developments, such

*For this work financial support has been received from Telenor and Leonardo Da Vinci
[†]University of Canterbury, New Zealand
[‡]Eindhoven University of Technology, The Netherlands
[§]SINTEF Industrial Management, Norway

as digital technology and architectures like ATM [1, 8], increasing the transportation capabilities of networks; new services, like Voice Mail, requiring more computational resources; and the computing industry's influence upon a deregulated telecommunication market. From the prognosis that the problem of allocating node resources will be important in near future (*c.f.* the Internet) we were asked by the financial contributor to examine the situation with sole focus on processing.

It is assumed that the transportation network, the telecom markets and the distributed processing capabilities of the network nodes are such that the following hold. Firstly, subservice demand may be considered independently of the services that generated it. Secondly, transportation and customer location need not be considered [9]. This paper concerns the problem with only one node to install subservices on, with a single constrained computing resource, *e.g.*, processing capacity.

This paper examines the situation where there is a fixed cost or revenue associated with installing a subservice. There are several ways of interpreting this fixed amount. One natural situation explaining both positive and negative values is where the service provider provides a subservice for a third party. In the case of low demand for the subservice, the service provider charges a fixed amount to install it. For high demand, he instead offers to pay a fixed amount to be able to offer the subservice. In both cases, profit is also gained from each fulfilled subservice request. In the remainder this is called a fixed charge, denoted c_j for subservice j, regardless of whether the fixed amount is positive or negative.

Demand for subservice j is given in terms of requested capacity units, d_j. Each unit of demand met for subservice j gains a profit of q_j. Each subservice uses a given amount of capacity just being available, independent of the demand met. This is called the *installation requirement* of the subservice, denoted by r_j for subservice j.

To formalize the model, there are n subservices with $r_j, d_j, q_j \in \mathbb{Z}_+$, $c_j \in \mathbb{Z}$, $j = 1, \ldots n$ and $S \in \mathbb{Z}_+$. The binary decision variables z_j indicate when subservice j is installed on the node, $z_j = 1$, or not, $z_j = 0$, $j = 1, \ldots, n$. The decision variables x_j give demand met for subservice j in node capacity units, $j = 1, \ldots, n$.

The objective is to maximize profit. The *capacity constraint* ensures the node capacity is not exceeded and the *installation constraints* require demand for a subservice to be met only if the subservice is installed.

Assumption 1 *For simplicity demands are pre-processed so that* $d_j \leq S - r_j$, $\forall j$.

The mathematical programming formulation of the fixed charge single node service provision problem (FSP) is given by

$$\begin{array}{lll} \max & \sum_{j=1}^{n} c_j z_j + \sum_{j=1}^{n} q_j x_j & \\ \text{s.t.} & \sum_{j=1}^{n} r_j z_j + \sum_{j=1}^{n} x_j \leq S, & \\ & d_j z_j \quad - x_j \geq 0 & j = 1, \ldots, n, \\ & z_j \in \{0,1\}, \quad x_j \geq 0 & j = 1, \ldots, n. \end{array} \tag{1}$$

This problem is important in its own right, and also has applications outside telecommunications. For example, it can be used in production planning on a single machine: the time available is S, there is a set-up time, r_j, on job j and no requirement that jobs are completed. Profit depends linearly on the processing time of each job, with a fixed cost, c_j, for starting job j.

The results from this paper also have implications for solution methods for the strongly NP-hard stochastic service provision problem where demand is uncertain. The problem addressed in this paper arises as a subproblem when a Lagrangian relaxation is used to decomposed into scenarios a two-stage version of the stochastic service provision problem without fixed charges. In this context a scenario represents one realisation of the stochastic demand variables. The decomposition scheme leads to solving many deterministic (FSP) problems.

Solution methods for other variants of the problem have been discussed in the literature. Feasibility of the service provision problem with multiple nodes and a requirement that all demand be met is described in [4] and shown to be strongly NP-complete. When there are no fixed charges the multiple node problem where profit is maximized is also strongly NP-hard [2], even for a constant number of nodes. For the single node problem without fixed charges (SP) (this is (FSP) with $c_j = 0$, $\forall j$), a fully polynomial time approximation scheme (fptas) exists [2]. The analysis of (SP) has many similarities with the knapsack problem [7]. The stochastic single node problem, with uncertain demand, is strongly NP-hard, unless the number of scenarios describing the probability distribution of the demands is fixed [3].

The contribution of this paper is to show that pseudo-polynomial time algorithms exist for (FSP). Section 2 discusses the LP-relaxation then Section 3 gives pseudo-polynomial dynamic programming algorithms for (FSP). A (fptas) based on DP and scaling is given in Section 4.

2 LP-relaxation

The LP relaxation (FSPLP) of (1), can be solved in $O(n)$ time and its optimal value is at most twice the optimal value of (1).

The LP-relaxation has many similarities with both continuous knapsack [7] and with the LP-relaxation, (SPLP), of (SP) [2]. In [2] it is shown that (SPLP) may be solved by little more than ordering the subservices by decreasing profit per unit capacity, $q_j d_j/(r_j + d_j)$, using the algorithm for the continuous knapsack problem with item price $q_j d_j$ and size $r_j + d_j$.

For (FSPLP), the algorithm needs an extension to allow for the possibility of installing a service but not meeting any demand when the fixed charge is favourable. The subservices are partitioned into three sets A, B and C. C contains subservices that will never be installed in an LP-solution because $c_j + q_j d_j \leq 0$. A contains subservices not in C where it is no more profitable per unit capacity to install the subservice without meeting any demand than it is to meet all demand, $c_j + q_j d_j > 0$ and $c_j/r_j \leq q_j$. B contains subservices for which it is more profitable per unit capacity to install the subservice without meeting any demand than it is to meet all demand for it. For $j \in B$ $c_j/r_j > q_j$ (> 0).

One set of knapsack items, A_0, is created from set A. For $j \in A$ there is a knapsack item in A_0 with size $r_j + d_j$ and price $c_j + q_j d_j$. Two sets of knapsack items, B_z and B_x, are created from set B. For $j \in B$ there is item $i_z \in B_z$ with size r_j and price c_j and item $i_x \in B_x$ with size d_j and price $q_j d_j$. For these subservices it is more profitable to increase z_j than x_j. The continuous knapsack problem with capacity S and items $A_0 \cup B_z \cup B_x$ is equivalent to (FSPLP). The number of knapsack items is at most $2n$.

An optimal solution to this continuous knapsack can be found adding the items by decreasing profit per unit capacity until either all items are added or critical item t is reached and fractionally added to fully use the capacity. Such a solution may be found in $O(n)$ time [7]. This solution corresponds to a solution to (FSPLP) in the following way. Any item fully added from A_0 or B_z has $z_j = 1$ for the corresponding subservice j. Any item added from A_0 or B_x has $x_j = d_j$ for the corresponding subservice j. When there is a critical item t, let l be the corresponding subservice. The node capacity will be fully with used with $0 < z_l = x_l/d_l < 1$, if $t \in A_0$, $0 < z_l < 1$ and $x_l = 0$, if $t \in B_z$, and $0 < x_l < d_l$, if $l \in B_x$ (in this case $z_l = 1$). Remaining subservices have $z_j = x_j = 0$.

Optimality of this solution follows from the definition of A, B and C. Notice that, when $t \in B_x$ the solution to (FSPLP) is feasible for (FSP). Such solutions are observed in computational experiments.

Let π^{LP} and π^{OPT} be the optimal value of (FSPLP) and (FSP), respectively.

$$\pi^{LP} = \sum_{j=1}^{l-1}(c_j + d_j) + c_l z_l + q_l x_l \leq 2\pi^{OPT}.$$

A greedy heuristic with performance ratio 2 follows directly from this, *c.f.* [7, 2].

3 Dynamic programming approach

For (FSP) it may be profitable to install subservices with no demand met.

Lemma 1 *There is an optimal solution to (FSP) with at most one fractional subservice, l, (called the critical subservice) with $0 < x_l < d_l$.*

Categorize the decision for subservice j as **D1**: $z_j = x_j = 0$, **D2**: $z_j = 1$ $x_j = d_j$, **D3**: $z_j = 1$ $x_j \in (0, d_j)$, **D4**: $z_j = 1$ $x_j = 0$. Any optimal solution, (z, x), with at most one critical demand can be described as a string $[D_1, ..., D_n]$ where at most one j has $D_j =$ D3 and $D_j \in \{$D1, D2, D3, D4$\}$ describes the decision category for subservice $j = 1, ..., n$.

Lemma 2 *When $q_1 \geq q_2 \geq \ldots \geq q_n$, there is an optimal solution where*

- *l is the index of a D3 decision, l is the first D4 decision or $l = n + 1$, and*
- *$D_j \in \{D1, D2\}$ for $j < l$, and $D_j \in \{D1, D4\}$ for $j > l$.*

A dynamic program for (FSP) based on Lemma 2 is now given. The subservices are sorted by decreasing q. The stages of the DP-formulation correspond to the subservices $1, ..., n$ and the states are (used) capacity. The recursion function at stage j, f_j gives the maximal profit obtainable from a state's capacity using the subservices $1, ..., j$. In addition the state space is extended by a parameter b indicating if a state is *flexible* ($b = 1$) or *inflexible* ($b = 0$) according to the following.

Definition 1 *A state at stage j is flexible if it has achieved its recursion function value with only D1 and D2 decisions up to and including stage j.*

Definition 2 *A state at stage j is inflexible if it has achieved its recursion function value by including a D3 or D4 decision at stage j or earlier.*

The state space is defined by pairs $(s, b) \in \{1, ..., S\} \times \{0, 1\}$ with $2S$ states. $f_j(s, 1)$ is the maximum profit achievable with capacity s using subservices $1, ..., j$ where $(s, 1)$ is a flexible state. $f_j(s, 0)$ is similar but $(s, 0)$ is an inflexible state. To simplify notation write $f_j(s, 1)$ as $F_j(s)$ and $f_j(s, 0)$ as $IF_j(s)$.

Starting from $F_0(0) = 0$, $F_j(s) = -\infty$, for $(s, j) \neq (0, 0)$, and $IF_0(0) = 0$, $IF_j(s) = -\infty$, for $(s, j) \neq (0, 0)$ leads, for $j = 1, ..., n$, to the recursions:

$$\begin{aligned} F_j(s) &= \max\{F_{j-1}(s - r_j - d_j) + c_j + q_j d_j, F_{j-1}(s)\} \\ IF_j(s) &= \max_{0 \leq x_j < d_j}\{F_{j-1}(s - r_j - x_j) + c_j + q_j x_j, IF_{j-1}(s - r_j) + c_j, IF_{j-1}(s)\}. \end{aligned}$$

For the $F_j(s)$, the two terms correspond to extending a flexible state at stage $j - 1$ with a D2 and D1 decision, respectively. For the $IF_j(s)$, the first term corresponds to extending a flexible state at stage $j - 1$ with a D3 or D4 decision while the last two terms correspond to extending an inflexible state at stage $j - 1$ with a D4 and D1 decision, respectively. The optimal solution is the highest of $F_n(S)$ and $IF_n(S)$.

The algorithm based on the above recursion is to generate:

```
Flexible to Inflexible Transition #1
procedure FlexInFlex(var $\Pi$, var $L, \Pi_F, L_F, j, q, c, r, d, S$)
begin
    for $i := 0$ to $S$ step 1
        $\Pi(i) := 0$;    $L(i) := \emptyset$;
    end for
    for $x := 0$ to $d - 1$ step 1
        for $i := r + x$ to $S$ step 1
            $new\pi := \Pi(i - r - k) + c + qx$
            if $\Pi(i) < new\pi$ and $\Pi_F(i) < new\pi$ then
                $\Pi(i) := new\pi$;    $L(i) := L(i - r - k) \cup \{j\}$;
            end if
        end for
    end for
end
```

Figure 1: Straightforward implementation of *FlexInFlex*.

1. *inflexible* states at stage j from *flexible* states at $j-1$, $D_j \in \{\mathrm{D3}, \mathrm{D4}\}$.
2. *inflexible* states at stage j from *inflexible* states at $j-1$. This is as with knapsack with item size r_j and profit c_j, $D_j \in \{\mathrm{D1}, \mathrm{D4}\}$.
3. *flexible* states at stage j from *flexible* states at $j-1$. As with knapsack with item size $r_j + d_j$ and profit $c_j + q_j d_j$, $D_j \in \{\mathrm{D1}, \mathrm{D2}\}$.

The straightforward way of generating inflexible states from flexible ones is to consider all $x_j \in \{0, \ldots, d_j - 1\}$ for every flexible state. Evaluating $F_j(s)$ and $IF_j(s)$ for $s = 0, \ldots, S$ with $j = 1, \ldots, n$ the time complexity is $O(nS^2)$. Storing all intermediate information has space complexity $O(nS)$.

Overall control of the three steps is straightforward. Steps 2 and 3 are as for knapsack. Improvements are most easily made in Step 1. The transition above is shown in Figure 1. Here, Π is the vector of new inflexible state profits, Π_F is the vector of old flexible state profits and L and L_F are lists of sets describing the corresponding solutions. j is the current stage (subservice), q, c, r and d are the problem parameters for this stage and S gives the node capacity (maximum state). var declares a parameter to act as input and output.

For the remainder of this section states at stage j are assumed to be flexible, while those at stage $j+1$ are assumed to be inflexible. State s at stage j can be used to generate states at stage $j+1$ between $\lambda(s) = s + r_j$ and and $\mu(s) = s + r_j + d_j - 1$.

Lemma 3 *$s_1 < s_2$ can both be used to generate s_{new} if and only if $s_{new} \in p(s_1, s_2) = \{\lambda(s_2), \lambda(s_2) + 1, \ldots, \mu(s_1) - 1, \mu(s_1)\}$. $|p(s_1, s_2)| = \max(0, d_j + s_1 - s_2) \leq d_j - 1$.*

Definition 3 *$s_1 < s_2$ are known as overlapping states when $|p(s_1, s_2)| > 0$.*

Lemma 4 *For two overlapping states of stage j, $s_1 < s_2$ it is only necessary to generate states in $p(s_1, s_2)$ at stage $j+1$ from s_1 if $F_j(s_2) < F_j(s_1) + (s_2 - s_1)q_j$ and from s_2 otherwise.*

The above can be utilized when the recursion table is calculated. In current state s_{cur} at stage j examine all decisions from $x_j \in \{\ell(s_{cur}), \ldots, d_j - 1\}$ where

```
Flexible to Inflexible Transition #2
procedure FlexInFlex(var $\Pi$, var $L, \Pi_F, L_F, j, q, c, r, d, S$)
begin
    for $i := 0$ to $S$ step 1
        $\ell(i) := 0$; $\Pi(i) := 0$; $L(i) := \emptyset$;
    end for
    $s_{cur} := 0$
    while $s_{cur} \leq S$ do
        for $x = \ell(s_{cur})$ to $d_j - 1$ step 1
            $s_{cand} := s_{cur} + x$;    $s_{new} := s_{cur} + r + x$;
            if $s_{new} > S$ then exit for
            if $\Pi_F(s_{cand}) \geq \Pi_F(s_{cur}) + qx$ and $s_{cand} <> s_{cur}$ then
                $\ell(s_{cand}) = UpdateList(s_{cand})$                                    *
                exit for
            else
                $new\pi := \Pi_F(s_{cur}) + c + qx$
                if $new\pi > \Pi(s_{new})$ and $new\pi > \Pi_F(s_{new})$ then
                    $\Pi(s_{new}) = new\pi$; $L(s_{new}) = L_F(s_{cur}) \cup j$; $\ell(s_{cand}) = d - x$;
                end if
                $\ell(s_{cand}) = UpdateList(s_{cand})$                                    *
            end if
        end for
        $s_{cur} = First()$                                                                *
    end while
end
```

Figure 2: A better transition from flexible to inflexible states

$\ell(s)$ is the lowest x_j to consider for the state s. Start with $\ell(s) = 0\ \forall s$. Each x decision leads to state $s_{new} = s_{cur} + r_j + x_j$ at stage $j + 1$. Also consider $s_{cand} = s_{cur} + x_j$ as a candidate to generate s_{new}. If s_{cand} generates s_{new} with a higher profit than s_{cur}, s_{cand} is preferred in all new states in $p(s_{cur}, s_{cand})$. Hence if $F_j(s_{cand}) \geq F_j(s_{cur}) + q_j x_j$, state $s_{cur} + 1$ can now be considered. Otherwise, update $\ell(s_{cand})$. The new $\ell(s_{cand})$ must satisfy $s_{cand} + r_j + \ell(s_{cand}) = \mu(s_{cur}) + 1$, *i.e.*, $\ell(s_{cand}) = d_j - x_j$. The computational complexity of the DP remains $O(nS^2)$. Preliminary computational results favour this transition.

The computational complexity can be improved by keeping track of a sorted *next-list* of non-dominated states as follows. When s_{cand} dominates s_{cur} jump directly to it. Otherwise, put s_{cand} as early as possible in the next-list, removing all states. The $\ell(s_{cand})$ is adjusted to reflect the fact that the first elements in the next-list are preferred. When s_{cur} is completely explored the next state explored is the first state in the next-list.

The next-list can be implemented as a circular array of length $d_j - 1$ with floating start and end. The overall computational complexity becomes $O(nS \log S)$. An algorithm based on this is given in Figure 2. Here, $UpdateList(s_{cand})$ inserts s_{cand} into the next-list, returning the new $\ell(s_{cand})$, and $First()$ returns the first state in the next-list, updating the start pointer. The changes from the first improvement of this section are the three lines marked *.

Preliminary computational tests carried out on 201 generated test problems indicated that the two improved DP algorithms out-perform standard branch and bound both in terms of average performance and in the variability of solution time.

4 Fully polynomial approximation scheme

Firstly a DP recursion is given with profit as the state space. A scaling approach is employed to produce a (fptas). The stages of the DP-formulation correspond to the subservices. Take π as a state of the recursion, this is the profit to be achieved. $F_j(\pi)$ is the minimum node capacity required to obtain a profit of at least π in a flexible state using the subservices $1,...,j$, $j = 1,...,n$. $IF_j(\pi)$ is the minimum node capacity required to obtain a profit of at least π in an inflexible state.

Starting with $F_0(0) = 0$, $F_j(\pi) = \infty$, $\forall(\pi, j) \neq (0,0)$, and $IF_0(0) = 0$, $IF_j(\pi) = \infty$, $\forall(\pi, j) \neq (0,0)$, leads, for $j = 1,...,n$, to the recursion

$$\begin{aligned} F_j(\pi) &= \min\{F_{j-1}(\pi - c_j - d_jq_j) + r_j + d_j, F_{j-1}(\pi)\} \\ IF_j(\pi) &= \min_{0\le x_j<d_j}\{F_{j-1}(\pi - c_j - x_jq_j) + r_j + x_j, IF_{j-1}(\pi - c_j) + r_j, IF_{j-1}(\pi)\}. \end{aligned}$$

The optimal solution is found in the nth stage.

Evaluating $F_j(\pi)$ and $IF_j(\pi)$ uses $d_j + 1$ comparisons. Evaluating $F_j(\pi)$ and $IF_j(\pi)$ for all $j = 1,...,n$ before any $F_j(\pi + 1)$ or $IF_j(\pi + 1)$, has time complexity $O(n\pi^{OPT}S)$. Storing all intermediate information has $O(n\pi^{OPT})$ space complexity.

The recursion can be improved as with the DP of Section 3 leading to a computational complexity of $O(n\pi^{OPT}\log S)$.

The following (fptas) is inspired by work by Ibarra and Kim [5], Lawler [6] and Dye, Stougie and Tomasgard [2]. The problem is reformulated with each variable x_j scaled by d_j to obtain variable $y_j = x_j/d_j$ with values in $\{0, 1/d_j, \ldots, (d_j-1)/d_j, 1\}$, $j = 1,...,n$. The profit coefficients are multiplied by the demand, $q'_j = q_jd_j$.

From this formulation, scale the profit coefficients with $k > 1$: $\tilde{q}_j = \lfloor\frac{q'_j}{k}\rfloor$ and $\tilde{c}_j = \lfloor\frac{c_j}{k}\rfloor$, $j = 1,...,n$. Use the above DP then multiply the solution value obtained by k as an approximation, $\tilde{\pi}$, to the π^{OPT}. Let y_j^{OPT} and $\tilde{y}_j$, $j = 1,...,n$, be the optimal solution for the original and the scaled problems, respectively.

$$\begin{aligned} \tilde{\pi} &= k(\sum_{j=1}^{n}\tilde{q}_j\tilde{y}_j + \sum_{j=1}^{n}\tilde{c}_j\tilde{z}_j) \ge k(\sum_{j=1}^{n}\tilde{q}_jy_j^{OPT} + \sum_{j=1}^{n}\tilde{c}_jz_j^{OPT}) \\ &\ge \sum_{j=1}^{n}(q'_j - k)y_j^{OPT} + \sum_{j=1}^{n}(c_j - k)z_j^{OPT} \ge \pi^{OPT} - k\sum_{j=1}^{n}(y_j^{OPT} + z_j^{OPT}). \end{aligned}$$

Taking $k = \epsilon\pi^G/2n$, where π^G is the solution value from the greedy heuristic mentioned in Section 2, it follows that

$$\frac{\pi^{OPT} - \tilde{\pi}}{\pi^{OPT}} \le \epsilon\frac{\sum_{j=1}^{n}(y_j^{OPT} + z_j^{OPT})}{2n}\frac{\pi^G}{\pi^{OPT}} \le \frac{\sum_{j=1}^{n}(y_j^{OPT} + z_j^{OPT})}{2n}\epsilon \le \epsilon.$$

Time complexity for the scaled recursion is $O(n\pi^{OPT}\log S/k) = O(n^2\log S/\epsilon)$ as $\pi^{OPT}/k \le 4n/\epsilon$. This can be improved by techniques like those suggested in [6].

5 Conclusions

This paper discusses applications and solution methods for the single node service provision problem with fixed charges. The problem has applications in telecommunications, where fixed charges come from the pricing structure, and in other areas like production planning with set-up costs. The fixed charges can come from dual prices in a scenario decomposition of the stochastic service provision problem without fixed charges.

The paper shows how the LP-relaxation may be solved in $O(n)$ time, giving theoretical motivation and computational results that indicate that it is worthwhile to solve the LP and check for integer feasibility. This may be particularly useful in a scenario decomposition approach for the stochastic service provision problem as many (FSP)'s are solved and the check may be achieved in $O(n)$ time.

A set of pseudo-polynomial time dynamic programming algorithms are given for the problem. Preliminary, computational results show that dynamic programming is a fast and robust approach to solve the problem to optimality. A fully polynomial time approximation scheme is also given.

The original motivation was to find a pseudo-polynomial algorithm to solve subproblems in a scenario decomposition of the stochastic version of the service provision problem. We have shown that such algorithms exist, but only more work can show if such decomposition schemes make it possible to solve hard stochastic problems. The motivation for this future work on decomposition is not speed, but being able to solve or bound problem instances that are too difficult for branch and bound with LP based bounds.

References

[1] M. De Prycker. *Asynchronous Transfer Mode, Solution for Broadband ISDN.* Prentice Hall, New Jersey, 1995.

[2] S. Dye, L. Stougie, and A. Tomasgard. Approximation algorithms and relaxations for a service provision problem on a telecommunication network. Working paper #2-98, Department of industrial economics and technology management, Norwegian university of science and technology, N-7034 Trondheim, Norway, 1998., 1998.

[3] S. Dye, L. Stougie, and A. Tomasgard. The stochastic single node service provision problem. Working paper #3-98, Department of industrial economics and technology management, Norwegian university of science and technology, N-7034 Trondheim, Norway, 1998, 1998.

[4] S. Dye, A. Tomasgard, and S.W. Wallace. Feasibility in transportation networks with supply eating arcs. *Networks*, 31:165–176, 1998.

[5] O.H. Ibarra and C.E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. Assoc. Comput. Mach.*, 22:463–468, 1975.

[6] E.L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.

[7] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & sons, 1990.

[8] R. Onvural. *Asynchronous Transfer Mode Networks: Performance Issues.* Artech House, Boston, 1994.

[9] Asgeir Tomasgard, Jan A. Audestad, S. Dye, Leen Stougie, Maarten H. van der Vlerk, and Stein W. Wallace. Modelling aspects of distributed processing in telecommunication networks. *Annals of Operations Research*, 82:161–184, 1998.