

---

## **AIMMS Modeling Guide - Telecommunication Network Problem**

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit [www.aimms.com](http://www.aimms.com) or order your hard-copy at [www.lulu.com/aimms](http://www.lulu.com/aimms).

Copyright © 1993–2011 by Paragon Decision Technology B.V. All rights reserved.

|                                  |                                  |                                  |
|----------------------------------|----------------------------------|----------------------------------|
| Paragon Decision Technology B.V. | Paragon Decision Technology Inc. | Paragon Decision Technology Pte. |
| Schipholweg 1                    | 500 108th Avenue NE              | Ltd.                             |
| 2034 LS Haarlem                  | Ste. # 1085                      | 80 Raffles Place                 |
| The Netherlands                  | Bellevue, WA 98004               | UOB Plaza 1, Level 36-01         |
| Tel.: +31 23 5511512             | USA                              | Singapore 048624                 |
| Fax: +31 23 5511517              | Tel.: +1 425 458 4024            | Tel.: +65 9640 4182              |
|                                  | Fax: +1 425 458 4025             |                                  |

Email: [info@aimms.com](mailto:info@aimms.com)  
WWW: [www.aimms.com](http://www.aimms.com)

AIMMS is a registered trademark of Paragon Decision Technology B.V. IBM ILOG CPLEX and sc CPLEX is a registered trademark of IBM Corporation. GUROBI is a registered trademark of Gurobi Optimization, Inc. KNITRO is a registered trademark of Ziena Optimization, Inc. XPRESS-MP is a registered trademark of FICO Fair Isaac Corporation. MOSEK is a registered trademark of Mosek ApS. WINDOWS and EXCEL are registered trademarks of Microsoft Corporation.  $\text{T}_{\text{E}}\text{X}$ ,  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ , and  $\text{A}_{\text{M}}\text{S}-\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  are trademarks of the American Mathematical Society. LUCIDA is a registered trademark of Bigelow & Holmes Inc. ACROBAT is a registered trademark of Adobe Systems Inc. Other brands and their products are trademarks of their respective holders.

Information in this document is subject to change without notice and does not represent a commitment on the part of Paragon Decision Technology B.V. The software described in this document is furnished under a license agreement and may only be used and copied in accordance with the terms of the agreement. The documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Paragon Decision Technology B.V.

**Paragon Decision Technology B.V. makes no representation or warranty with respect to the adequacy of this documentation or the programs which it describes for any particular purpose or with respect to its adequacy to produce any particular result. In no event shall Paragon Decision Technology B.V., its employees, its contractors or the authors of this documentation be liable for special, direct, indirect or consequential damages, losses, costs, charges, claims, demands, or claims for lost profits, fees or expenses of any nature or kind.**

**In addition to the foregoing, users should recognize that all complex software systems and their documentation contain errors and omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors shall not be responsible under any circumstances for providing information or corrections to errors and omissions discovered at any time in this book or the software it describes, whether or not they are aware of the errors or omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors do not recommend the use of the software described in this book for applications in which errors or omissions could threaten life, injury or significant loss.**

This documentation was typeset by Paragon Decision Technology B.V. using  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  and the LUCIDA font family.

## Chapter 21

# A Telecommunication Network Problem

In this chapter you will encounter a capacity utilization problem in a telecommunication network. Traffic in such a network is expressed in terms of calls, and calls are made between a large number of origin-destination pairs during a particular period of time. Calls between origins and destinations can be routed along any path through the network subject to capacity limitations. The objective is to identify bottleneck capacity in the network. In practical applications, the model turns out to be quite large due to the many possible paths that exist between origins and destinations. For that reason a path generating technique is introduced to control the size of the model that is passed to a solver during a sequence of iterations.

*This chapter*

The telecommunication network model discussed in this chapter can be found in various sources. Two references, spanning a period of almost 30 years, are [Hu69] and [Me98]. The required theory of linear programming and column generation can be found in [Ch83] and in Chapter 19 of this book. In addition, Chapter 20, 'A Cutting Stock Problem', also provides an application in which column generation plays a central role.

*References*

Linear Program, Network Program, Simplex Method, Column Generation, Auxiliary Model, Customized Algorithm, Mathematical Derivation, Worked Example.

*Keywords*

---

### 21.1 Problem description

This section provides a brief introduction to the terminology and concepts used in the telecommunication network problem described in this chapter. The problem itself is summarized towards the end of this section.

*This section*

In a telecommunication network, transmission lines are used to carry traffic in the form of calls. These lines form the link between switch-stations. Traffic is routed from one switch-station to the next until it reaches its destination. For the sake of simplicity both the origin and destination of a call are assumed to be switch-stations.

*Network configuration*

Each switch-station is represented as a *node* in the network, and each link between any two nodes is represented as an *arc*. The maximum amount of traffic that can go through a switch-station during a specific period of time will be referred to as *node capacity*. A similar definition holds for *arc capacity*. A route from origin to destination through the network is a *path*.

*Nodes, arcs and paths*

Traffic for a particular origin-destination pair can be split and subsequently recombined at any node in the network. This flexibility in routing traffic allows for efficient use of the entire network. The implication of flexible routing for the model to be developed, is that all possible paths between an origin and a destination will need to be considered.

*Flexible routing*

Assume that the amount of traffic between all origin-destination pairs for a particular period is known, and that the capacity for all switch-stations and transmission lines is provided. The problem that will be addressed is the bottleneck identification problem. In this problem traffic is routed along paths in the network so that traffic requirements are met. In addition, the bottleneck in the network is identified by finding that arc or node with the largest percentage use of the available capacity.

*Problem summary*

The bottleneck identification problem can be viewed as a strategic network design problem. In a network there are often bottlenecks that must be alleviated through redesign either by adding new switch-stations or new transmission lines, or by adding capacity to any of the existing facilities. The problem in this chapter represents a simplification, because it does not consider such practical matters as network robustness and reliability under (uncertain) traffic regimes. Nevertheless, the model developed next is of interest, as it forms a basis for several extensions.

*Network design*

---

## 21.2 Bottleneck identification model

In this section you will encounter a compact arc-path formulation of the bottleneck identification problem described in the previous section. Initially, it is assumed that all possible paths between origin-destination pairs are enumerated explicitly. This assumption will be relaxed in the next section where paths are generated one-at-a-time as needed.

*This section*

Figure 21.1 depicts a simplified Dutch telecommunication network containing 6 nodes and 12 (bi-directional) arcs. In this example, it is assumed that there is traffic between all possible pairs of (origin-destination) nodes, and that each node and arc has a limited capacity. Even in this small example, the number of undirected paths is quite large (namely 377), and only a few of them are listed in Table 21.1.

*Example*

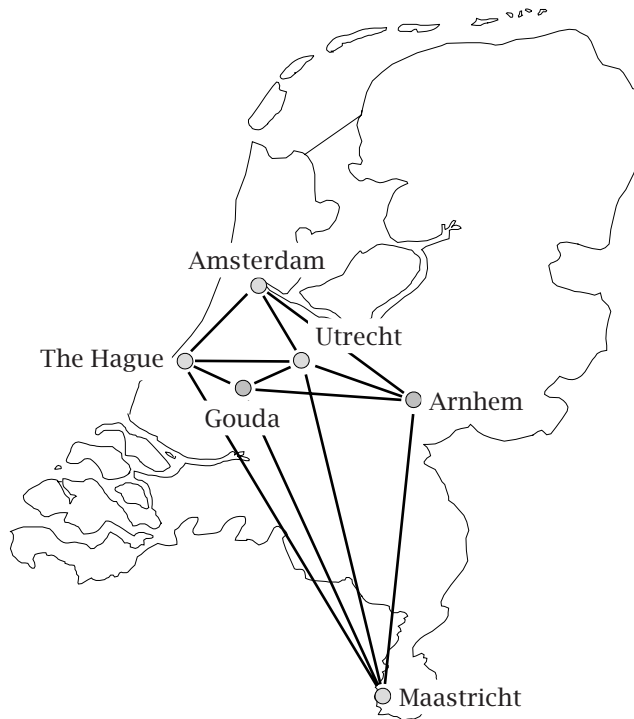


Figure 21.1: A Dutch Telecommunication Network

The verbal model is expressed in terms of network terminology for reasons of conciseness and ease of recall. The interpretation in terms of transmission lines and switch-stations is straightforward. Note that the use of any arc or node (as referred to in the problem summary) is expressed as a fraction of the available capacity.

*Verbal model description*

**Minimize:** *maximum fraction of either arc use or node use,*

**Subject to:**

- *for each origin-destination pair: total traffic along paths connecting this pair is equal to the required amount of traffic,*
- *for each arc in the network: total traffic that uses that arc is equal to a fraction of the available arc capacity,*
- *for each node in the network: total traffic that uses that node is equal to a fraction of the available node capacity,*
- *for each arc in the network: capacity use is less than or equal to the maximum fraction,*
- *for each node in the network: capacity use is less than or equal to the maximum fraction.*

The mathematical description is slightly more complicated due to the various indices that play a role.

| origin    | destination | path  |
|-----------|-------------|---|
| Amsterdam | Maastricht  | Amsterdam - The Hague - Maastricht                            |
| -         | -           | Amsterdam - The Hague - Utrecht - Maastricht                  |
| -         | -           | Amsterdam - The Hague - Utrecht - Gouda - Maastricht          |
| -         | -           | Amsterdam - The Hague - Utrecht - Gouda - Arnhem - Maastricht |
| -         | -           | Amsterdam - The Hague - Utrecht - Arnhem - Maastricht         |
| -         | -           | Amsterdam - The Hague - Utrecht - Arnhem - Gouda - Maastricht |
| -         | -           | Amsterdam - The Hague - Gouda - Maastricht                    |
| -         | -           | Amsterdam - The Hague - Gouda - Utrecht - Maastricht          |
| -         | -           | Amsterdam - The Hague - Gouda - Utrecht - Arnhem - Maastricht |
| -         | -           | Amsterdam - The Hague - Gouda - Arnhem - Maastricht           |
| -         | -           | Amsterdam - The Hague - Gouda - Arnhem - Utrecht - Maastricht |
| -         | -           | Amsterdam - Utrecht - Maastricht                              |
| -         | -           | Amsterdam - Utrecht - The Hague - Maastricht                  |
| ...       |             |   |

Table 21.1: Paths between Amsterdam and Maastricht

The following symbols will be used for the mathematical description of the bottleneck identification model.

*Mathematical description*

**Indices:**

|        |                                     |
|--------|-------------------------------------|
| $n$    | <i>nodes</i>                        |
| $a$    | <i>arcs</i>                         |
| $o, d$ | <i>origin and destination nodes</i> |
| $p$    | <i>paths</i>                        |

**Set:**

|          |   |
|----------|---|
| $S_{od}$ | <i>all paths between origin <math>o</math> and destination <math>d</math></i> |
|----------|---|

**Parameters:**

|          |  |
|----------|--|
| $A_{ap}$ | <i>incidence: arc <math>a</math> is on path <math>p</math></i>                       |
| $B_{np}$ | <i>incidence: node <math>n</math> is on path <math>p</math></i>                      |
| $C_a$    | <i>capacity of arc <math>a</math></i>  |
| $C_n$    | <i>capacity of node <math>n</math></i>   |
| $D_{od}$ | <i>required traffic between origin <math>o</math> and destination <math>d</math></i> |

**Variables:**

|       |  |
|-------|--|
| $x_p$ | <i>traffic along path <math>p</math></i>                     |
| $f_a$ | <i>fraction of available capacity of arc <math>a</math></i>  |
| $f_n$ | <i>fraction of available capacity of node <math>n</math></i> |
| $M$   | <i>maximum fraction of either arc or node capacity</i>       |

Traffic requirements are typically specified for only a subset of all origin-destination pairs. Such traffic may be routed along any path. As the set  $S_{od}$  contains all paths for each  $(o, d)$  pair, specifying the corresponding traffic requirement is straightforward.

*Traffic requirement*

$$\sum_{p \in S_{od}} x_p = D_{od} \quad \forall (o, d) \mid D_{od} > 0$$

Note that the above traffic requirement constraint is only defined when the required amount of traffic  $D_{od}$  is greater than zero. Enforcing this condition is one way to reduce the number of constraints. A second and practically more effective reduction in the number of constraints is to add the requirements  $D_{od}$  and  $D_{do}$ , and to consider only traffic requirement constraints for  $o < d$ .

*Reducing the number of constraints*

An arc  $a$  can be on several paths that are selected to meet the traffic requirement for various  $(o, d)$  pairs. The total (bi-directional) amount of traffic along such an arc, however, is limited by the arc's capacity. Rather than specifying a hard capacity constraint for each arc, an additional nonnegative variable  $f_a$  is introduced to indicate the fraction of capacity used for that arc. Such a fraction should, of course, be less than or equal to one. By leaving it unrestricted from above, however, it measures the fraction of capacity sufficient to meet traffic requirements.

*Arc capacity utilization*

$$\sum_p A_{ap} x_p = f_a C_a \quad \forall a \mid C_a > 0$$

Note that the above arc capacity utilization constraint is only defined when the corresponding arc capacity  $C_a$  is greater than zero.

A node  $n$  can also be on several paths selected to meet traffic requirements for various  $(o, d)$  pairs. Just like arcs, nodes also have limited capacity. An additional nonnegative variable  $f_n$  is introduced to measure the fraction of capacity sufficient to meet traffic requirements.

*Node capacity utilization*

$$\sum_p B_{np} x_p = f_n C_n \quad \forall n \mid C_n > 0$$

Note that the above node capacity utilization constraint is only defined when the corresponding node capacity  $C_n$  is greater than zero.

Identifying the bottleneck capacity is now straightforward to model. Consider the maximum capacity utilization fraction  $M$ , and let all fractions of capacity utilization in the network be less than or equal to this fraction. By minimizing  $M$ , the optimal solution will identify one or more critical capacities. Note that the underlying linear program will always be feasible, because there is no effective capacity limitation (i.e. limit on  $M$ ).

*Identifying bottleneck capacity*

$$\begin{aligned} \text{Minimize:} & \quad M \\ \text{Subject to:} & \quad f_a \leq M \quad \forall a \mid C_a > 0 \\ & \quad f_n \leq M \quad \forall n \mid C_n > 0 \end{aligned}$$

An optimal  $M \leq 1$  indicates that existing capacities can be utilized to meet all traffic requirements. An optimal  $M > 1$  implies that capacity expansion is needed.

The following mathematical statement summarizes the model.

*Model summary*

**Minimize:**

$M$

**Subject to:**

$$\begin{aligned} \sum_{p \in S_{od}} x_p &= D_{od} & \forall (o, d) \mid o < d, D_{od} > 0 \\ \sum_p A_{ap} x_p &= f_a C_a & \forall a \mid C_a > 0 \\ \sum_p B_{np} x_p &= f_n C_n & \forall n \mid C_n > 0 \\ 0 &\leq x_p & \forall p \\ 0 &\leq f_a \leq M & \forall a \mid C_a > 0 \\ 0 &\leq f_n \leq M & \forall n \mid C_n > 0 \end{aligned}$$

---

### 21.3 Path generation technique

The number of paths in practical applications is too large to enumerate, and thus the corresponding linear program cannot be generated in its entirety. This section develops a path generating scheme which avoids the complete enumeration of all paths. The approach resembles the column generating technique described in Chapter 20, except that the columns now correspond to paths.

*This section*

If all columns of a linear program cannot be generated prior to solving, they need to be generated 'as needed' during the solution phase. This requires insight into the construction of a column together with knowledge of the relevant shadow prices to make sure that any generated column will improve the objective function value of the underlying linear program. In the case of the bottleneck identification model, you need to consider the path-related variable  $x_p$  and see whether there is a new column  $p$  that may lead to a better solution of the overall model.

*Dynamic column generation*

Each new column with coefficients corresponding to a new  $x_p$  variable has zero-one entries in the first three symbolic constraints of the above bottleneck identification model. The entries associated with the first constraint are all 0 except for a single 1 corresponding to a particular  $(o, d)$  pair. The entries of the new column associated with the second constraint are 1 or 0 dependent on whether an arc is on the new path or not. Let  $z_a$  be 1 if arc  $a$  is on the path, and 0 otherwise. Similarly, the entries associated with the third constraint are 1 or 0 dependent on whether a node is on the new path or not. Let  $h_n$  be 1 if node  $n$  is on the path, and 0 otherwise.

*A typical column*

The symbols  $z_a$  and  $h_n$  can be viewed as variables characterizing the path-related coefficients of each new column to be determined. The shadow prices corresponding to the first three constraints are needed to decide on a new column to be added to the bottleneck identification model. They are:

*Column entry condition ...*

$$\begin{aligned}\lambda_{od} & \text{ for the traffic requirement constraint} \\ \mu_a & \text{ for the arc capacity utilization constraint} \\ \theta_n & \text{ for the node capacity utilization constraint}\end{aligned}$$

You may want to verify that the condition below describes the situation in which a path  $p$  may contribute to the optimal solution value of the underlying bottleneck identification model. This condition, which is equivalent to the column entry condition in the simplex method for a minimization model, is similar to the reduced cost criterion explained in Chapters 19 and 20.

$$0 - \lambda_{od} - \sum_a \mu_a z_a - \sum_n \theta_n h_n < 0$$

By considering only those values of  $z_a$  and  $h_n$  that together determine a path, and by minimizing the left side of the above inequality over all particular  $(o, d)$  pairs, you obtain the best path in terms of the reduced cost criterion. If the minimum value is strictly less than zero, you have found a path between an  $o$  and a  $d$  that will improve the underlying linear program. If this minimum is greater than or equal to zero, then there does not exist a new path that will improve the linear program. This observation leads to the idea to employ a path-finding model as an auxiliary model to identify a new path  $p$  satisfying the above column entry condition. In this chapter a path-finding linear programming formulation has been selected. A shortest-path approach based on Dijkstra's algorithm could also have been employed.

*... leads to minimization*

The following auxiliary model is selected to play a role in the path generating approach proposed in this chapter. The notation in this auxiliary path-finding model is based on the concept of directed arcs for reason of convenience. Its relationship to the column entry condition above will be discussed in subsequent paragraphs.

*Auxiliary path-finding model*

**Sets:**

$$\begin{aligned}N & \text{ nodes} \\ I \subset N & \text{ intermediate nodes}\end{aligned}$$

**Indices:**

$$i, j \quad \text{nodes}$$

**Element Parameters:**

$$\begin{aligned}orig & \text{ originating node} \\ dest & \text{ destination node}\end{aligned}$$

**Numerical Parameter:**

$$k_{ij} \quad \text{objective coefficient for arc } (i, j)$$

**Variable:**

$y_{ij}$             1 if arc  $(i, j)$  is on optimal path, 0 otherwise

When  $k_{ij} > 0$  and the set  $I$  contains all nodes except the originating node *orig* and the destination node *dest*, you may verify that the following model with network constraints determines a best path from *orig* to *dest* expressed in terms of indices  $i$  and  $j$ .

**Minimize:**

$$\sum_{(ij)} k_{ij} y_{ij}$$

**Subject to:**

$$\sum_j y_{ji} - \sum_j y_{ij} = \begin{cases} -1 & \text{if } i = \text{orig} \\ 0 & \text{if } i \in I \\ 1 & \text{if } i = \text{dest} \end{cases}$$

The above model can be solved using a specialized algorithm. It can also be solved as a linear program with constraints  $0 \leq y_{ij} \leq 1$ , which will result in a zero-one optimal solution due to the network property of the model (see e.g. Chapter 5).

Not all  $y_{ij}$  variables need to be considered in the above model formulation. First of all, all variables with  $i = j$  are superfluous, as no such variable will be in the optimal solution with  $k_{ii} > 0$ . Similarly, all variables  $y_{ij}$  with  $i = \text{dest}$  or with  $j = \text{orig}$  are also superfluous. As a result, the number of relevant  $y_{ij}$  variables is equal to  $|N|(|N| - 3) + 3$  for  $|N| \geq 2$ . Throughout the remainder of this section the restricted domain of  $y_{ij}$  is implied.

*Restricting the  $y_{ij}$  domain*

It is not immediately obvious how the above auxiliary path-finding model can be used to minimize the expression

*Required translation*

$$-\lambda_{od} - \sum_a \mu_a z_a - \sum_n \theta_n h_n$$

The required translation turns out to be fairly straightforward, but demands some attention. In essence, the  $z$  and  $h$  terms must be translated into the  $y_{ij}$  decision variables, while the other terms must be translated into the objective function coefficients  $k_{ij}$ .

The term  $\lambda_{od}$  is a constant once a particular  $(o, d)$  pair is selected. Therefore, this term can be ignored when constructing the objective function coefficients of the auxiliary model for a particular pair.

*The  $\lambda$  terms*

An arc  $a$  corresponds to a tuple  $(i, j)$  as well as a tuple  $(j, i)$ . Therefore, a possible translation is to write  $z_a = y_{ij} + y_{ji}$  with  $\mu_{ij} = \mu_{ji} = \mu_a$ . Such a translation is permitted, because at most one of the  $y_{ij}$  and  $y_{ji}$  values can be equal to 1 in an optimal solution of the auxiliary model when  $k_{ij} > 0$ .

*The  $\mu$  and  $z$  terms*

Let  $(o, d) = (\text{orig}, \text{dest})$ . You can express the relationship between  $h_n$  and  $y_{ij}$  as follows. First of all,  $h_o = \sum_j y_{oj}$  and  $h_d = \sum_i y_{id}$ . Then for all intermediate nodes, either  $h_{i \in I} = \sum_j y_{ij}$  or  $h_{i \in I} = \sum_j y_{ji}$ . The term  $\theta_n$  needs not be modified, and can be used directly in the construction of the objective function coefficients  $k_{ij}$ .

*The  $\theta$  and  $h$  terms*

The column entry condition without the constant term  $\lambda_{od}$  is

*Rewriting the column entry condition*

$$-\sum_a \mu_a z_a - \sum_n \theta_n h_n$$

and can now be rewritten in terms of the  $y_{ij}$  variables of the auxiliary model in one of two ways depending on the expression for  $h_{i \in I}$ . Either

$$-\sum_{(ij)} \mu_{ij} y_{ij} - \theta_o \sum_j y_{oj} - \sum_{i \in I} \theta_i \sum_j y_{ij} - \theta_d \sum_i y_{id}$$

or

$$-\sum_{(ij)} \mu_{ij} y_{ij} - \theta_o \sum_j y_{oj} - \sum_{i \in I} \theta_i \sum_j y_{ji} - \theta_d \sum_i y_{id}$$

By carefully combining terms in the above two expressions and considering only the restricted  $(i, j)$  domain, the corresponding values of  $k_{ij}$  can be written either as

*Determining the coefficients  $k_{ij}$*

$$\begin{aligned} k_{oj} &:= -\mu_{oj} - \theta_o && \forall j \neq d \\ k_{ij} &:= -\mu_{ij} - \theta_i && \forall (i, j), i \neq o, j \neq d \\ k_{id} &:= -\mu_{id} - \theta_i - \theta_d && \forall i \end{aligned}$$

or as

$$\begin{aligned} k_{oj} &:= -\mu_{oj} - \theta_j - \theta_o && \forall j \\ k_{ij} &:= -\mu_{ij} - \theta_i && \forall (i, j), i \neq o, j \neq d \\ k_{id} &:= -\mu_{id} - \theta_d && \forall i \neq o \end{aligned}$$

The values  $\mu$  and  $\theta$  are typically zero when the corresponding capacity constraints in the bottleneck identification model are not critical. Once a capacity constraint has an associated capacity fraction value equal to the critical value  $M$ , then the corresponding  $\mu$  or  $\theta$  value will be strictly less than 0, causing the corresponding  $k_{ij}$  to be greater than 0. You may verify this by applying the definition of a shadow price as described in Section 4.2 to the capacity constraints after moving all variable terms to the left-hand side. By adding a sufficiently small  $\epsilon > 0$  to all permitted values of  $k_{ij}$ , the requirement of  $k_{ij} > 0$  is automatically satisfied and the optimal solution of the auxiliary path finding model is guaranteed to be a proper path between  $o$  and  $d$  without any

*Forcing  $k_{ij} > 0$*

zero-valued subtours. A recommended choice for  $\epsilon$  is the smallest positive initial  $k_{ij}$  value divided by the total number of positive initial  $k_{ij}$  values.

Once the optimal solution of the auxiliary path finding model has been determined, a check must be made to verify whether the newly generated path should be added to the bottleneck identification model. This check is nothing more than verifying whether the column entry condition, described in terms of the  $y_{ij}$  values, is satisfied. You may verify that the following expression forms the correct check.

*Correcting  $k_{ij}$   
afterwards*

$$-\lambda_{od} + \sum_{ij} (k_{ij} - \epsilon) y_{ij} < 0$$

Recall that the values of  $k_{ij}$  could have been determined in one of two ways, but both sets are appropriate when checking the above condition.

The translation of the column entry condition into the terminology of the auxiliary path-finding model is now complete. The following algorithmic skeleton loosely summarizes the approach to solve the bottleneck identification model with path generation. Initialization (without initial shadow prices) is accomplished by setting the objective function coefficients  $k_{ij}$  equal to 1, leading to the computation of the shortest path with the fewest number of intermediate nodes between every possible  $(o, d)$  pair. These paths determine the initial entries of the parameters  $A_{ap}$  and  $B_{np}$  in the bottleneck identification model (referred to as 'main model' below). In addition, each such shortest path is also a single initial element of the set  $S_{od}$ .

*Path generation  
algorithmic  
skeleton*

```

FOR all origin-destination pairs DO
  Solve path-finding model
  Add shortest path to parameters in main model
ENDFOR

WHILE at least one new path has been added DO
  Solve current bottleneck model
  FOR all origin-destination pairs DO
    Solve path-finding model
    IF new path contributes
      THEN add path to parameters in main model
    ENDFOR
  ENDFOR
ENDWHILE

```

An implementation of this algorithm in AIMMS is not entirely straightforward, and does require some carefully constructed set and parameter manipulations to update the input of both the path-finding model and the bottleneck identification model. Special care is also required when implementing the column entry condition in order to avoid the repeated generation of a single path. The powerful language features of AIMMS, however, allow for a one-to-one translation of the notation used in this chapter.

In the above proposed algorithm a new path between all possible  $(o, d)$  pairs is computed prior to solving the next bottleneck capacity model. By restricting yourself to only those  $(o, d)$  pairs with paths along critical nodes and/or arcs (i.e. nodes and/or arcs with maximum capacity utilization fractions), the computational effort to find new paths prior to solving the next bottleneck identification model can be significantly reduced. The suggestion in this paragraph is only one of several possibilities to reduce computational effort.

*Reducing  
computational  
efforts*

## 21.4 A worked example

The traffic requirements (in terms of calls) between all possible origins and destination pairs are presented in Table 21.2.

*Traffic data*

| $o$        | $d$ | Amsterdam | Utrecht | The Hague | Gouda | Arnhem | Maastricht |
|------------|-----|-----------|---------|-----------|-------|--------|------------|
| Amsterdam  |     |           | 55      | 95        | 20    | 30     | 45         |
| Utrecht    | 90  |           |         | 50        | 10    | 15     | 20         |
| The Hague  | 85  | 45        |         |           | 15    | 10     | 30         |
| Gouda      | 35  | 25        | 35      |           |       | 10     | 15         |
| Arnhem     | 45  | 15        | 20      | 5         |       |        | 35         |
| Maastricht | 60  | 25        | 40      | 10        | 30    |        |            |

Table 21.2: Traffic requirements between all origin-destination pairs

The node and arc capacities for the network in this example are provided in Table 21.3.

*Capacity data*

|            | Arc capacities |         |           |       |        |            | Node capacities |
|------------|----------------|---------|-----------|-------|--------|------------|-----------------|
|            | Amsterdam      | Utrecht | The Hague | Gouda | Arnhem | Maastricht |                 |
| Amsterdam  |                | 360     | 300       |       | 240    |            | 490             |
| Utrecht    |                |         | 360       | 60    | 90     | 120        | 340             |
| The Hague  |                |         |           | 90    |        | 180        | 400             |
| Gouda      |                |         |           |       | 40     | 120        | 220             |
| Arnhem     |                |         |           |       |        | 210        | 280             |
| Maastricht |                |         |           |       |        |            | 340             |

Table 21.3: Arc and node capacities in the network

The initial number of shortest paths between all possible  $(o, d)$  pairs is 15 ( $= \binom{6}{2}$ ). These paths were generated by the shortest-path procedure, and are summarized in Table 21.4. Note that only the arc-path incidences are provided. An explicit path description in terms of nodes can always be derived from the arc names.

*Initial paths*

| Arcs                   | Initial paths |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------|---------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|                        | 01            | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 |
| (Amsterdam,Utrecht)    | ×             |    | ×  |    | ×  |    |    |    |    |    |    |    |    |    |    |
| (Amsterdam,The Hague)  |               | ×  |    |    |    |    |    |    |    |    |    | ×  |    |    |    |
| (Amsterdam,Arnhem)     |               |    |    | ×  |    |    |    |    |    |    |    | ×  |    |    |    |
| (Utrecht,The Hague)    |               |    |    |    |    | ×  |    |    |    |    |    |    |    |    |    |
| (Utrecht,Gouda)        |               |    | ×  |    |    |    | ×  |    |    |    |    |    |    |    |    |
| (Utrecht,Arnhem)       |               |    |    |    |    |    |    | ×  |    |    |    |    |    |    |    |
| (Utrecht,Maastricht)   |               |    |    |    | ×  |    |    |    | ×  |    |    |    |    |    |    |
| (The Hague,Gouda)      |               |    |    |    |    |    |    |    |    | ×  |    |    |    |    |    |
| (The Hague,Maastricht) |               |    |    |    |    |    |    |    |    |    |    | ×  |    |    |    |
| (Gouda,Arnhem)         |               |    |    |    |    |    |    |    |    |    |    |    | ×  |    |    |
| (Gouda,Maastricht)     |               |    |    |    |    |    |    |    |    |    |    |    |    | ×  |    |
| (Arnhem,Maastricht)    |               |    |    |    |    |    |    |    |    |    |    |    |    |    | ×  |

Table 21.4: Initial paths generated by shortest-path algorithm

After continuing the path generating procedure, a total of 9 additional paths were generated. The observed bottleneck fraction was reduced from 1.500 (based on the initial paths only) to a value of 1.143 (based on both the initial and additional paths). The 9 additional paths are summarized in Table 21.5. It is of interest to note that the bottleneck identification model was solved 5 times in total to obtain these results. The number of times the auxiliary path-finding model was solved, amounted to 90.

*Additional paths*

| Arcs                   | Additional paths |    |    |    |    |    |    |    |    |  |  |   |  |
|------------------------|------------------|----|----|----|----|----|----|----|----|--|--|---|--|
|                        | 16               | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |  |  |   |  |
| (Amsterdam,Utrecht)    |                  | ×  |    |    |    |    |    |    |    |  |  |   |  |
| (Amsterdam,The Hague)  |                  | ×  | ×  | ×  |    |    |    |    |    |  |  |   |  |
| (Amsterdam,Arnhem)     | ×                |    |    |    |    |    |    | ×  | ×  |  |  |   |  |
| (Utrecht,The Hague)    |                  |    |    |    |    |    |    |    |    |  |  |   |  |
| (Utrecht,Gouda)        |                  |    |    |    |    |    | ×  |    |    |  |  |   |  |
| (Utrecht,Arnhem)       |                  |    |    |    |    |    |    | ×  |    |  |  |   |  |
| (Utrecht,Maastricht)   |                  |    |    |    |    |    |    |    |    |  |  |   |  |
| (The Hague,Gouda)      |                  |    | ×  |    |    |    |    |    |    |  |  | × |  |
| (The Hague,Maastricht) |                  |    |    | ×  | ×  |    | ×  |    |    |  |  |   |  |
| (Gouda,Arnhem)         | ×                |    |    |    |    |    |    |    |    |  |  |   |  |
| (Gouda,Maastricht)     |                  |    | ×  |    | ×  | ×  | ×  |    | ×  |  |  |   |  |
| (Arnhem,Maastricht)    |                  |    |    |    | ×  |    | ×  | ×  | ×  |  |  |   |  |

Table 21.5: Additional paths generated by path-finding algorithm

When the algorithm was modified and restricted to path generation for critical  $(o, d)$  pairs only, 4 instead of 9 additional new paths were generated. In this case, the bottleneck identification model was solved 4 instead of 5 times, and the auxiliary path-finding model was solved 35 instead of 90 times. As expected, the bottleneck fraction was again reduced to 1.143.

*Restricting to critical paths*

---

## 21.5 Summary

In this chapter you have encountered a bottleneck identification problem in a telecommunication network, together with two model formulations. The first formulation assumes that the input to the model is based on all possible paths between origins and destinations. The second (more practical) formulation does not explicitly enumerate all possible paths, but generates them only when they can contribute to the identification of the bottleneck capacity. An auxiliary model is used to generate these paths. The overall model can be used to decide how to modify existing capacity to meet changing traffic requirements, or how existing traffic is to be routed through the network.

---

## Exercises

- 21.1 Consider the path generation technique presented in Section 21.3 and implement the bottleneck identification model of Section 21.2 using the example data contained in Tables 21.2 and 21.3. Verify whether the optimal solution found with AIMMS coincides with the one presented in Tables 21.4 and 21.5.
- 21.2 Investigate whether there is any difference in the optimal solution due to the choice of the  $k_{ij}$  coefficient values developed in Section 21.3.
- 21.3 Adjust your search algorithm to examine only those  $(o, d)$  pairs with paths along critical nodes and/or arcs. Verify for yourself how much the amount of computational work has been reduced due to this modification.

## Bibliography

- [Ch83] V. Chvátal, *Linear programming*, W.H. Freeman and Company, New York, 1983.
- [Hu69] T.C. Hu, *Integer programming and network flows*, Addison-Wesley, Reading, Massachusetts, 1969.
- [Me98] E.A. Medova, *Chance-constrained stochastic programming for integrated services network management*, *Annals of Operations Research* **81** (1998).