Universität Würzburg
Institut für Informatik
Research Report Series

Fast heuristics for optimal routing in large IP networks

LTM Berry, S. Köhler, D Staehle and P Tran-Gia

Report No. 262   July 2000

Lehrstuhl für Informatik III,
Universität Würzburg
Am Hubland,
97074 Würzburg
Germany
E-Mail: koehler@informatik.uni-wuerzburg.de

Abstract

We consider the problem of computing optimal sets of paths between source-destination pairs in large IP networks. Given the traffic demands between the pairs of nodes, link capacities and the topological structure of the network we wish to allocate the traffic streams to routes such that either the maximum link utilisation is minimized, or the average link utilisation of the network is minimized. An interesting feature of the optimisation is that the set of paths selected, for all pairs, must be constrained to satisfy shortest path principles.
Two approaches are developed. The first, based on use of k'th shortest paths, iteratively modifies link costs to achieve the selected objective. The second method seeks to apply network decomposition effectively such that the LP technique of Staehle, Köhler and Kohlhaas [11] may be implemented on each subgraph in an iterative algorithm.

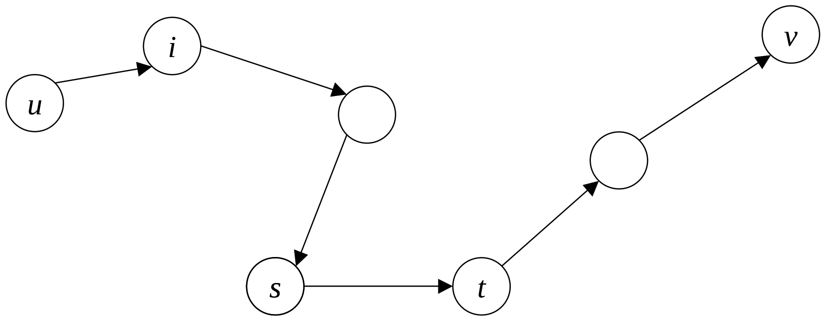
1. Introduction

The problem considered in this research report is described fully in [11].
Essentially, given a fixed network defined by its link adjacency matrix, the traffic demands between node pairs of the network and link capacities, an optimal routing pattern is required. The routing pattern (set of paths between the source-destination pairs) is optimal if the carried flows meet the link capacity constraints, minimize an objective function and satisfy shortest path principles. That is, once the paths have been determined costs (not necessarily unique) can be assigned to the links of the network such that application of a shortest (cheapest) path algorithm to the network would result in the unique selection of the previously mentioned optimal routing pattern. The objectives considered in [10] are to minimize the maximum link utilisation, or to minimize the average network link utilisation or to minimize a linear combination of these two values.
To solve the first problem, that is to find the optimal path allocations, the problem is formulated as an Integer Program. Because only one path is to be selected for each source-destination pair (an "all or nothing assignment") and the selected paths must correspond to shortest paths, these routing constraints imply that the solution to the integer program can be found using Linear Programming (LP). That is, the solution to the LP gives integer values for the decision variables.
The method for finding a set of link costs that would generate the optimal paths using shortest path routing at each node is to also formulate and solve a second LP. This problem is an inversion of the Dijkstra [3] shortest path problem. In this report we will focus on the first part of the problem, that is, the determination of the optimal set of paths.
At the present time, finding an optimal solution by the two-LP method of [11] seems to be restricted to problems with less than 30 or so nodes.

The number of constraint rows in the LP is the limiting factor.
In order to extend the range of applicability for large networks and to improve the computational speed we consider two heuristic approaches. The first, based on use of *k*'th shortest paths, iteratively modifies link costs to achieve the selected objective. The second method seeks to apply network decomposition effectively such that the LP technique of Staehle, Köhler and Kohlhaas [11] may be implemented on each subgraph in an iterative algorithm.
We first define the optimal routing problem and describe the LP formulation.

2. Formulation of the Problem

The decision variables are zero-one variables x_{ij}^{uv}, which take the value 1 if and only if the link *(i,j)* belongs to the selected path for source-destination pair *u-v*. The traffic demand from *u* to *v* is f_{uv} and the capacity of link (*i,j*) is c_{ij}. The ratio of the total flow on link (*i,j*) to its capacity c_{ij} gives the link's utilization.
The objective function (see below) comprises two terms, the second being the sum of all link utilisations for the entire network (this is proportional to the average link utilisation) and the first is a bound on the maximum link utilisation. Minimizing the variable *t* (a percentage, see the utilisation constraints below) effects the reduction of the maximum link utilisation. The parameter a_t is a weight to control the importance of the first term relative to the second term.
The routing constraints deserve special comment.
The insistence on shortest path routing means that if the path from node *i* to node *v* includes link (*s,t*) then if the path from *u* to *v* passes through node *i*, it also must include link (*s,t*).



The derivation of the routing constraints is illustrated below.
Other constraints in the formulation are standard network constraints.

$x_{ui}^{uv} = 1 \Rightarrow x_{st}^{iv} \le x_{st}^{uv}$ can be formulated as

$x_{st}^{iv} + Mx_{ui}^{uv} \le M + x_{st}^{uv}$ where M is an upper bound for

$x_{st}^{iv} - x_{st}^{uv}$ as these are 0-1 variables we can take $M = 1$

Thus $x_{st}^{iv} + x_{ui}^{uv} \le 1 + x_{st}^{uv}$ or $x_{st}^{iv} + x_{ui}^{uv} - 1 \le x_{st}^{uv}$

Formulation

Minimise $a_t t + \sum_{ij} \sum_{uv} \frac{f_{uv} x_{ij}^{uv}}{c_{ij}}$

Transport Constraints

$\sum_{j=1, c_{ij}>0}^{N} x_{ij}^{uv} \le 1$ for all $u-v$ (for each u-v at most one link out of router i may carry f^{uv})

$\sum_{i=1, c_{ui}>0}^{N} x_{ui}^{uv} = 1$ (exactly one link from the source u carries flow)

$\sum_{j=1, c_{ij}>0}^{N} x_{ij}^{uv} - \sum_{j=1, c_{ji}>0}^{N} x_{ji}^{uv} = 0$ (conservation equation - if there is one incoming link to node i

carrying u-v flow then there is exactly 1 outgoing link)

$\sum_{i=1, c_{iv}>0}^{N} x_{iv}^{uv} = 1$ (exactly one link into router v carries u - v flow)

Capacity Constraints

$$\sum_{uv} x_{ij}^{uv} f^{uv} \le a_c c_{ij} \qquad \textit{for all links } (i, j)$$

Utilisation Constraints

$$\sum_{uv} x_{ij}^{uv} f^{uv} \le \frac{tc_{ij}}{100}$$

Routing Constraints

$$x_{ui}^{uv} + x_{st}^{iv} - x_{st}^{uv} \le 1 \qquad \textit{for all flows } u\text{-}v, \textit{ routers } i \notin \{u,v\} \textit{ and links } (s,t)$$

$$x_{jv}^{uv} + x_{st}^{uj} - x_{st}^{uv} \le 1 \qquad \textit{for all flows } u\text{-}v, \textit{ routers } j \notin \{u,v\} \textit{ and links } (s,t)$$

$$(i.e.\ x_{ui}^{uv} = 1 \Rightarrow x_{st}^{iv} \le x_{st}^{uv} \quad and \quad x_{jv}^{uv} = 1 \Rightarrow x_{st}^{uj} \le x_{st}^{uv})$$

Delay Constraints

$$\sum_{ij} x_{ij}^{uv} d_{ij} \le a_r d_{\min}^{uv}$$

As the graph is undirected, to avoid cycles from the source (or destination) to an adjacent node and back to the same node (i.e. the existence of disconnected paths) we set the variables x_{iu}^{uv} and x_{vi}^{uv} equal to zero. The minimization of the average utilisation in the objective function will result in the avoidance of cycles in the connected paths between a source and destination.
The delay constraints specify that the delay on any selected path between nodes *u-v* must not exceed a multiple a_r of the minimum delay path $d_{\min}^{uv}$ between *u* and *v*.

3 *K* Shortest Path Cost Adjustment Heuristic

We reformulate the above problem directly in terms of link costs. Denoting the cost label on link (*i,j*) by w_{ij}, and the vector of link costs by $\underline{w}$ the formulation is:

$$Min \sum_{ij}\sum_{uv}\frac{f^{uv}x_{ij}^{uv}(\underset{\sim}{w})}{c_{ij}} + a_t t(\underset{\sim}{w})$$

$$\underset{\sim}{w} \geq \underset{\sim}{0}$$

That is, the objective function is simply a function of the link costs $\underset{\sim}{w}$.
We are given the network topology, the flow demands, link capacities and delay constraints. Once the link costs are specified the x_{ij}^{uv} follow directly from the shortest paths.
We will describe the heuristic with the aid of an example.
The first step is to allocate link costs to all links, and to then determine *k*'th shortest paths for each source-destination pair *u-v*. From the shortest paths we can compute the total load carried on each link and the link utilisations. The *k* shortest paths are used in re-allocating flows, that is, a re-selection of new shortest paths is made with the aim of reducing the objective function. The important question is, "How can we alter the link costs in an iterative algorithm such that the maximum link utilisation (and as far as possible also the average link utilization) are minim ized?" The details of how this is done are given in section 3.1
We first recall that in [11] it was necessary (the second LP) to determine a set of link costs consistent with a given set of shortest (cheapest) paths. This is a non-trivial inversion of the shortest path problem. The approach adopted in the heuristic avoids this inversion problem.

3.1 Example of the KSP Cost Adjustment Heuristic

We select the initial set of link costs such that links with large capacities have less cost than links with smaller capacities. Specifically the initial cost associated with link (*i,j*) is:

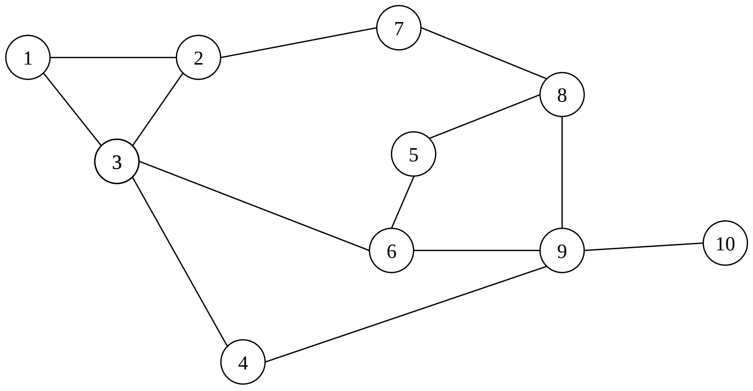
$[a\frac{c_{\max}}{c_{ij}}]$ where $c_{\max}$ is the maximum link capacity for the network,

c_{ij} is the capacity of link (i, j)

$[x]$ denotes the smallest integer $\geq$ x

and a is an integer multiplier.

We consider the following 10 node network:



The traffic demands between the nodes of the network and the link capacities are s shown in the two matrices below. Our objective is to determine a set of link costs such that under shortest path routing a minimization of the maximum link utilisation is achieved (as the prime objective) and also a reduction in the average link utilisation.

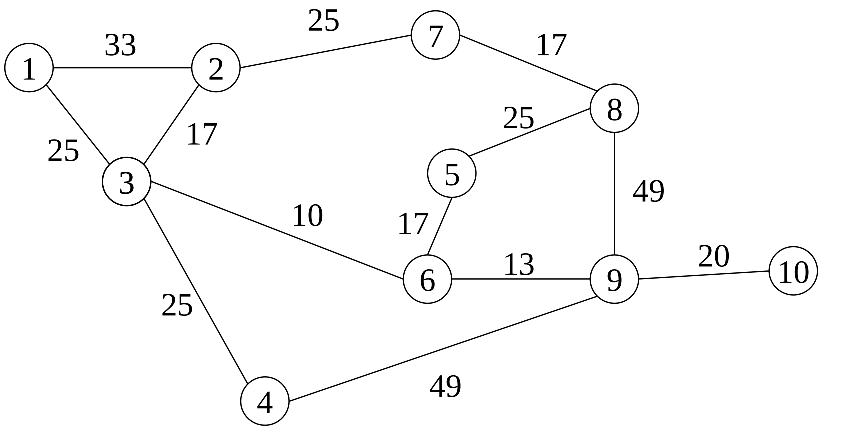
	1	2	3	4	5	6	7	8	9	10
1	0	8	5	12	3	7	4	10	6	1
2	8	0	2	11	9	14	3	13	5	7
3	5	2	0	3	7	6	4	3	14	8
4	12	11	3	0	4	1	2	1	7	3
5	3	9	7	4	0	3	7	2	9	5
6	7	14	6	1	3	0	8	10	4	2
7	4	3	4	2	7	8	0	7	2	1
8	10	13	3	1	2	10	7	0	3	9
9	6	5	14	7	9	4	2	3	0	4
10	1	7	8	3	5	2	1	9	4	0

Flow demands

	1	2	3	4	5	6	7	8	9	10
1	0	30	40							
2	30	0	60				40			
3	40	60	0	40		98				
4			40	0					20	
5					0	60		40		
6			98		60	0			80	
7		40					0	60		
8					40		60	0	20	
9				20		80		20	0	50
10									50	0

Link capacities

The initial set of link costs are now :



E.g. for link (1,2) its cost is [10*98/30] = 33

Initial link costs-   with *a = 10*

Next, we compute the *k*'th shortest paths for each source-destination pair. Yen's algorithm [12] was used for this purpose. It is described in the Appendix, together with some observations concerning computation of *k*'th shortest paths.
A feature of this heuristic is that it is only necessary to compute the *k*'th shortest paths once. Path cost adjustments are performed in a simple manner, thus avoiding an O(n^3) calculation at each iteration of the algorithm. For our example, the *k*=3 shortest paths are shown in the tables below. It is thought that it is only necessary to

consider a few shortest paths for each source-destination pair of nodes rather than to generate all paths satisfying the path delay constraints. Further experimentation will establish further information on what value of *k* to use; this will be a function of the size of a network.

P_{1-2}	1,2	33
P_{1-2}	1,3,2	42
P_{1-2}	1,3,6,5,8,7,2	119
P_{1-3}	1,3	25
P_{1-3}	1,2,3	50
P_{1-3}	1,2,7,8,5,6,3	127
P_{1-4}	1,3,4	50
P_{1-4}	1,2,3,4	75
P_{1-4}	1,3,6,9,4	97
P_{1-5}	1,3,6,5	52
P_{1-5}	1,2,3,6,5	77
P_{1-5}	1,2,7,8,5	100
P_{1-6}	1,3,6	35
P_{1-6}	1,2,3,6	60
P_{1-6}	1,3,4,9,6	112
P_{1-7}	1,2,7	58
P_{1-7}	1,3,2,7	67
P_{1-7}	1,3,6,5,8,7	94
P_{1-8}	1,2,7,8	75
P_{1-8}	1,3,6,5,8	77
P_{1-8}	1,3,2,7,8	84

P_{1-9}	1,3,6,9	48
P_{1-9}	1,2,3,6,9	73
P_{1-9}	1,3,4,9	99
P_{1-10}	1,3,6,9,10	68
P_{1-10}	1,2,3,6,9,10	93
P_{1-10}	1,3,4,9,10	119
P_{2-3}	2,3	17
P_{2-3}	2,1,3	58
P_{2-3}	2,7,8,5,6,3	94
P_{2-4}	2,3,4	42
P_{2-4}	2,1,3,4	83
P_{2-4}	2,3,6,9,4	89
P_{2-5}	2,3,6,5	44
P_{2-5}	2,7,8,5	67
P_{2-5}	2,1,3,6,5	85
P_{2-6}	2,3,6	27
P_{2-6}	2,1,3,6	68
P_{2-6}	2,7,8,5,6	84
P_{2-7}	2,7	25
P_{2-7}	2,3,6,5,8,7	86
P_{2-7}	2,3,6,9,8,7	106

P_{2-8}	2,7,8	42
P_{2-8}	2,3,6,5,8	69
P_{2-8}	2,3,6,9,8	89
P_{2-9}	2,3,6,9	40
P_{2-9}	2,1,3,6,9	81
P_{2-9}	2,7,8,9	91
P_{2-10}	2,3,6,9,10	60
P_{2-10}	2,1,3,6,9,10	101
P_{2-10}	2,7,8,9,10	111
P_{3-4}	3,4	25
P_{3-4}	3,6,9,4	72
P_{3-4}	3,6,5,8,9,4	150
P_{3-5}	3,6,5	27
P_{3-5}	3,2,7,8,5	84
P_{3-5}	3,6,9,8,5	97
P_{3-6}	3,6	10
P_{3-6}	3,4,9,6	87
P_{3-6}	3,2,7,8,5,6	101
P_{3-7}	3,2,7	42
P_{3-7}	3,6,5,8,7	69
P_{3-7}	3,1,2,7	83

P_{8-9}	8,9	49
P_{8-9}	8,5,6,9	55
P_{8-9}	8,7,2,3,6,9	82
P_{8-10}	8,9,10	69
P_{8-10}	8,5,6,9,10	75
P_{8-10}	8,7,2,3,6,9,10	102
P_{9-10}	9,10	20
P_{9-10}	-	-
P_{9-10}	-	-

P_{3-8}	3,6,5,8	52
P_{3-8}	3,2,7,8	59
P_{3-8}	3,6,9,8	72
P_{3-9}	3,6,9	23
P_{3-9}	3,4,9	74
P_{3-9}	3,6,5,8,9	101
P_{3-10}	3,6,9,10	43
P_{3-10}	3,4,9,10	94
P_{3-10}	3,6,5,8,9,10	121
P_{4-5}	4,3,6,5	52
P_{4-5}	4,9,6,5	79
P_{4-5}	4,3,2,7,8,5	109
P_{4-6}	4,3,6	35
P_{4-6}	4,9,6	62
P_{4-6}	4,3,2,7,8,5,6	126
P_{4-7}	4,3,2,7	67
P_{4-7}	4,3,6,5,8,7	94
P_{4-7}	4,3,1,2,7	108
P_{4-8}	4,3,6,5,8	77
P_{4-8}	4,3,2,7,8	84
P_{4-8}	4,3,6,9,8	97

P_{4-9}	4,3,6,9	48
P_{4-9}	4,9	49
P_{4-9}	4,3,6,5,8,9	126
P_{4-10}	4,3,6,9,10	68
P_{4-10}	4,9,10	69
P_{4-10}	4,3,6,5,8,9,10	146
P_{5-6}	5,6	17
P_{5-6}	5,8,9,6	87
P_{5-6}	5,8,7,2,3,6	94
P_{5-7}	5,8,7	42
P_{5-7}	5,6,3,2,7	69
P_{5-7}	5,6,9,8,7	96
P_{5-8}	5,8	25
P_{5-8}	5,6,9,8	79
P_{5-8}	5,6,3,2,7,8	86
P_{5-9}	5,6,9	30
P_{5-9}	5,8,9	74
P_{5-9}	5,6,3,4,9	101
P_{5-10}	5,6,9,10	50
P_{5-10}	5,8,9,10	94
P_{5-10}	5,6,3,4,9,10	121

P_{6-7}	6,3,2,7	52
P_{6-7}	6,5,8,7	59
P_{6-7}	6,9,8,7	79
P_{6-8}	6,5,8	42
P_{6-8}	6,9,8	62
P_{6-8}	6,3,2,7,8	69
P_{6-9}	6,9	13
P_{6-9}	6,3,4,9	84
P_{6-9}	6,5,8,9	91
P_{6-10}	6,9,10	33
P_{6-10}	6,3,4,9,10	104
P_{6-10}	6,5,8,9,10	111
P_{7-8}	7,8	17
P_{7-8}	7,2,3,6,5,8	94
P_{7-8}	7,2,3,6,9,8	114
P_{7-9}	7,2,3,6,9	65
P_{7-9}	7,8,9	66
P_{7-9}	7,8,5,6,9	72
P_{7-10}	7,2,3,6,9,10	85
P_{7-10}	7,8,9,10	86
P_{7-10}	7,8,5,6,9,10	92

Having found the shortest paths, we next compute the total flow on each link and their utilisations.

Link	Flow	Capacity	Utilisation (decreasing order)
3,6	117	98	1,19387755
2,7	47	40	1,175
3,4	44	40	1,1
2,3	62	60	1,03333333
6,9	74	80	0,925
5,6	54	60	0,9
1,3	34	40	0,85
9,10	40	50	0,8
1,2	22	30	0,73333333
7,8	37	60	0,61666667
8,9	12	20	0,6
5,8	23	40	0,575
4,9	0	20	0

Average utilisation	0,80786238
Maximum utilisation	1,19387755

Note: As the flow demand and link capacity matrices are symmetrical in this example, the link (6,3) has the same utilisation as link (3,6) etc.

CPLEX solution

Average utilisation	0.82894427
Maximum utilisation	0.96666666

We note that the initial feasible solution has a lower average utilisation for the network than that of the optimal solution found by the method described in [11], but the maximum utilisation is greater. We also note that the link with the greatest utilisation is link (3,6). Our initial strategy is to increase the cost of link (3,6). In order to describe the manner in which this is done, we introduce some notation.

P^{u-v} is the shortest path between u - v

$P^{u-v}(k)$ is the k'th shortest path between u - v $\quad (P^{u-v}(1) \equiv P^{u-v})$

$Q(i,j) = \{P^{u-v} : (i,j) \in P^{u-v}\}$ i.e. the set of shortest paths that use link (i, j)

d_{ij}^{u-v} is the smallest value of k such that link $(i, j) \notin P^{u-v}(k)$

$w(P^{u-v}(k))$ is the sum of the link costs for path $P^{u-v}(k)$.

i.e. $w(P^{u-v}(k)) = \sum_{(i,j)\in P^{u-v}(k)} m_{ij}$;where m_{ij} is the link cost for link (i, j)

$\Delta_{ij}^{u-v} = w(P^{u-v}(d_{ij}^{u-v})) - w(P^{u-v})$ i.e. the increment in the cost of link (i, j) needed to equalise path costs for $P^{u-v}(d_{ij}^{u-v})$ and P^{u-v}

For each *u-v*, we identify the next cheapest path that does not contain link (3,6) (the max. utilization link) and compute its cost difference from that of the cheapest path, $\Delta_{ij}^{u-v} = w(P^{u-v}(d_{ij}^{u-v})) - w(P^{u-v})$.

$u-v$	$d_{3,6}^{\text{u-v}}$	$\Delta_{3,6}^{\text{u-v}}$	f^{uv}	$P^{u-v}(d_{ij}^{u-v})$
1-5	3	48	3	1,2,7,8,5
1-6	3	77	7	1,3,4,9,6
1-9	3	51	6	1,3,4,9
1-10	3	51	1	1,3,4,9,10
2-5	2	23	9	2,7,8,5
2-6	3	57	14	2,7,8,5,6
2-9	3	51	5	2,7,8,9
2-10	3	51	7	2,7,8,9,10
3-5	2	57	7	3,2,7,8,5
3-6	2	77	6	3,4,9,6
3-8	2	7	3	3,2,7,8
3-9	2	51	14	3,4,9
3-10	2	51	8	3,4,9,10
4-5	2	27	4	4,9,6,5
4-6	2	27	1	4,9,6
4-8	2	7	1	4,3,2,7,8
4-9	2	1	7	4,9
4-10	2	1	3	4,9,10
7-9	2	1	2	7,8,9
7-10	2	1	1	7,8,9,10
7-6*	2	7	8	7,8,5,6

First find paths with least $\Delta_{3,6}^{\text{u-v}}$

i>j $\longrightarrow$

The source-destination pairs with the smallest cost difference are identified. In this example, the smallest value of Δ_{ij}^{u-v} is equal to 1 for pairs 4-9, 4-10, 7-9 and 7-10.

By incrementing the cost of link (3,6) by 2 we remove the traffic load for these source-destination pairs from link (3,6); thus decreasing its utilization. Formally, the next steps of the algorithm can be described as follows:

Denote the selected link with maximum utilisation by (i^*, j^*).

For our example this is link (3,6).

We identify paths $P^{u-v}(d_{i^*j^*}^{u-v})$ for which

$$\Delta_{i^*j^*}^{u-v} = w(P^{u-v}(d_{i^*j^*}^{u-v})) - w(P^{u-v}) = \min_{uv} \Delta_{i^*j^*}^{u-v} = \Delta_{\min}$$

The *next* shortest (cheapest) path between *u-v* **not** using link *(i*, j*)*

The difference in path costs

(Note that $w(P^{u-v}(d_{i^*j^*}^{u-v})) - w(P^{u-v}) \geq 0$)

The paths satisfying the above condition are: $P^{4-9}(2)$, $P^{4-10}(2)$, $P^{7-9}(2)$ and $P^{7-10}(2)$

Next we increment the cost for link (i^*, j^*)

$$m_{i^*j^*} = m_{i^*j^*} + \Delta_{\min} + 1$$

For our example, $m_{3,6} = m_{3,6} + 2 = 12$

The effect of all this is to shift flow away from link (3,6) onto links of the new cheapest paths: P^{4-9}, P^{4-10}, P^{7-9} and P^{7-10}

Thus the utilisation of link (3,6) will decrease

u-v	old path	new path	offered flow
4-9	4,3,6,9	4,9	7
4-10	4,3,6,9,10	4,9,10	3
7-9	7,2,3,6,9	7,8,9	2
7-10	7,2,3,6,9,10	7,8,9,10	1

↑ Paths containing link (3,6)

↑ Paths not containing link (3,6)

Link	Change in flow	New flow
(3,6)*	-7-3-2-1 = -13	104
(4,3)*	-7 -3 = -10	34
(6,9)	-7-3-2-1 = -13	61
(9,10)	3+3-1+1 = 0	40
(7,2)*	-2-1 = -3	44
(2,3)	-2-1 = -3	59
(4,9)	+7+3 = +10	10
(7,8)	+2+1 = +3	40
(8,9)	+2+1 = +3	15

Link (4,9) now receives flow- previously it was unused

* similar flow values for links (6,3), (3,4) and (2,7)

Results after one iteration of the KSP cost adjustment heuristic

Link	Flow	Capacity	Utilisation (decreasing)
3,6	117	98	1,19387755
2,7	47	40	1,175
3,4	44	40	1,1
2,3	62	60	1,03333333
6,9	74	80	0,925
5,6	54	60	0,9
1,3	34	40	0,85
9,10	40	50	0,8
1,2	22	30	0,73333333
7,8	37	60	0,61666667
8,9	12	20	0,6
5,8	23	40	0,575
4,9	0	20	0

Average utilisation 0,80786238

Maximum utilisation 1,19387755 (link (3,6))

Link	New flow	New utilisation
3,6	104	1,06122449
2,7	44	1,1
3,4	34	0,85
2,3	59	0,98333333
6,9	61	0,7625
5,6	54	0,9
1,3	34	0,85
9,10	40	0,8
1,2	22	0,73333333
7,8	40	0,66666667
8,9	15	0,75
5,8	23	0,575
4,9	10	

Average utilisation 0,77169676

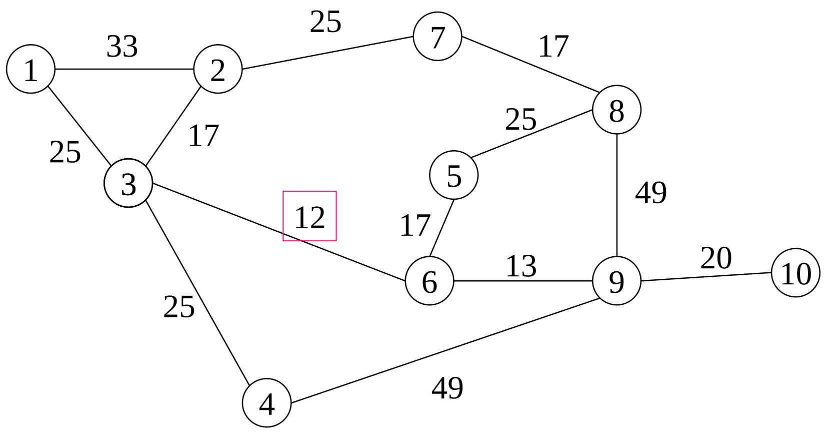
Maximum utilisation 1,1 (link (2,7)

CPLEX:

0.8289443

0.9666666

The link costs for the start of the second iteration are as shown below.



New link costs

A simple adjustment to the path costs, for those links of paths affected by the increase in cost to link (3,6) gives the new ordering of path costs shown in the following tables. The four cheapest paths are tabulated for each source-destination pair.

P_{1-2}	1,2	33
P_{1-2}(2)	1,3,2	42
P_{1-2}(3)	1,3,6,5,8,7,2	**121**
P_{1-2}(4)	1,3,6,9,8,7,2	**141**
P_{1-3}	1,3	25
P_{1-3}	1,2,3	50
P_{1-3}	1,2,7,8,5,6,3	**129**
P_{1-3}	1,2,7,8,9,6,3	**149**
P_{1-4}	1,3,4	50
P_{1-4}	1,2,3,4	75
P_{1-4}	1,3,6,9,4	**99**
P_{1-4}	1,2,3,6,9,4	**124**
P_{1-5}	1,3,6,5	**54**
P_{1-5}	1,2,3,6,5	**79**
P_{1-5}	1,2,7,8,5	100
P_{1-5}	1,3,2,7,8,5	109
P_{1-6}	1,3,6	35
P_{1-6}	1,2,3,6	**62**
P_{1-6}	1,3,4,9,6	112
P_{1-6}	1,2,7,8,5,6	117
P_{1-7}	1,2,7	58
P_{1-7}	1,3,2,7	67
P_{1-7}	1,3,6,5,8,7	**96**
P_{1-7}	1,3,6,9,8,7	**116**
P_{1-8}	1,2,7,8	75
P_{1-8}	1,3,6,5,8	**79**
P_{1-8}	1,3,2,7,8	84
P_{1-8}	1,3,6,9,8	**99**

P_{1-9}	1,3,6,9	**50**
P_{1-9}	1,2,3,6,9	**75**
P_{1-9}	1,3,4,9	99
P_{1-9}	1,2,3,4,9	124
P_{1-10}	1,3,6,9,10	**70**
P_{1-10}	1,2,3,6,9,10	**95**
P_{1-10}	1,3,4,9,10	119
P_{1-10}	1,2,3,4,9,10	144
P_{2-3}	2,3	17
P_{2-3}	2,1,3	58
P_{2-3}	2,7,8,5,6,3	**96**
P_{2-3}	2,7,8,9,6,3	**116**
P_{2-4}	2,3,4	42
P_{2-4}	2,1,3,4	83
P_{2-4}	2,3,6,9,4	**91**
P_{2-4}	2,7,8,5,6,3,4	**121**
P_{2-5}	2,3,6,5	**46**
P_{2-5}	2,7,8,5	67
P_{2-5}	2,1,3,6,5	**87**
P_{2-5}	2,3,6,9,8,5	**116**
P_{2-6}	2,3,6	**29**
P_{2-6}	2,1,3,6	**70**
P_{2-6}	2,7,8,5,6	84
P_{2-6}	2,7,8,9,6	104
P_{2-7}	2,7	25
P_{2-7}	2,3,6,5,8,7	**88**
P_{2-7}	2,3,6,9,8,7	**108**
P_{2-7}	2,1,3,6,5,8,7	**129**

P_{2-8}	2,7,8	42
P_{2-8}	2,3,6,5,8	**71**
P_{2-8}	2,3,6,9,8	**91**
P_{2-8}	2,1,3,6,5,8	**112**
P_{2-9}	2,3,6,9	**42**
P_{2-9}	2,1,3,6,9	**83**
P_{2-9}	2,7,8,9	91
P_{2-9}	2,3,4,9	91
P_{2-10}	2,3,6,9,10	**62**
P_{2-10}	2,1,3,6,9,10	**103**
P_{2-10}	2,7,8,9,10	111
P_{2-10}	2,3,4,9,10	111
P_{3-4}	3,4	25
P_{3-4}	3,6,9,4	**74**
P_{3-4}	3,6,5,8,9,4	**152**
P_{3-4}	3,2,7,8, 9,4	157
P_{3-5}	3,6,5	**29**
P_{3-5}	3,2,7,8,5	84
P_{3-5}	3,6,9,8,5	**99**
P_{3-5}	3,4,9,6,4	**104**
P_{3-6}	3,6	**12**
P_{3-6}	3,4,9,6	87
P_{3-6}	3,2,7,8,5,6	101
P_{3-6}	3,2,7,8,9,6	121
P_{3-7}	3,2,7	42
P_{3-7}	3,6,5,8,7	**71**
P_{3-7}	3,1,2,7	83
P_{3-7}	3,6,9,8,7	**91**

P_{3-8}	3,6,5,8	**54**
P_{3-8}	3,2,7,8	59
P_{3-8}	3,6,9,8	**74**
P_{3-8}	3,1,2,7,8	100
P_{3-9}	3,6,9	**25**
P_{3-9}	3,4,9	74
P_{3-9}	3,6,5,8,9	**103**
P_{3-9}	3,2,7,8,9	108
P_{3-10}	3,6,9,10	**45**
P_{3-10}	3,4,9,10	94
P_{3-10}	3,6,5,8,9,10	**123**
P_{3-10}	3,2,7,8,9,10	128
P_{4-5}	4,3,6,5	**54**
P_{4-5}	4,9,6,5	79
P_{4-5}	4,3,2,7,8,5	109
P_{4-5}	4,9,8,5	123
P_{4-6}	4,3,6	**37**
P_{4-6}	4,9,6	62
P_{4-6}	4,3,2,7,8,5,6	126
P_{4-6}	4,9,8,5,6	140
P_{4-7}	4,3,2,7	67
P_{4-7}	4,3,6,5,8,7	**96**
P_{4-7}	4,3,1,2,7	108
P_{4-7}	4,9,8,7	115
P_{4-8}	4,3,6,5,8	**79**
P_{4-8}	4,3,2,7,8	84
P_{4-8}	**4,9,8**	**98**
P_{4-8}	**4,3,6,9,8**	**99**

P_{4-9}	**4,9**	49
P_{4-9}	**4,3,6,9**	**50**
P_{4-9}	4,3,6,5,8,9	**128**
P_{4-9}	4,3,2,7,8,9	**133**
P_{4-10}	**4,9,10**	69
P_{4-10}	**4,3,6,9,10**	**70**
P_{4-10}	4,3,6,5,8,9,10	**148**
P_{410}	4,3,2,7,8,9,10	153
P_{5-6}	5,6	17
P_{5-6}	5,8,9,6	87
P_{5-6}	5,8,7,2,3,6	**96**
P_{5-6}	5,8,7,2,1,3,6	**137**
P_{5-7}	5,8,7	42
P_{5-7}	5,6,3,2,7	**71**
P_{5-7}	5,6,9,8,7	96
P_{5-7}	5,6,3,1,2,7	**112**
P_{5-8}	5,8	25
P_{5-8}	5,6,9,8	79
P_{5-8}	5,6,3,2,7,8	**88**
P_{5-8}	5,6,3,1,2,7,8	**129**
P_{5-9}	5,6,9	30
P_{5-9}	5,8,9	74
P_{5-9}	5,6,3,4,9	**103**
P_{5-9}	5,8,7,2,3,6,9	**109**
P_{5-10}	5,6,9,10	50
P_{5-10}	5,8,9,10	94
P_{5-10}	5,6,3,4,9,10	**123**
P_{5-10}	5,8,7,2,3,6,9,10	**129**

P_{6-7}	6,3,2,7	**54**
P_{6-7}	6,5,8,7	59
P_{6-7}	6,9,8,7	79
P_{6-7}	6,3,1,2,7	**95**
P_{6-8}	6,5,8	42
P_{6-8}	6,9,8	62
P_{6-8}	6,3,2,7,8	**71**
P_{6-8}	6,3,1,2,7,8	**112**
P_{6-9}	6,9	13
P_{6-9}	6,3,4,9	**86**
P_{6-9}	6,5,8,9	91
P_{6-9}	6,3,2,7,8,9	**120**
P_{6-10}	6,9,10	33
P_{6-10}	6,3,4,9,10	**106**
P_{6-10}	6,5,8,9,10	111
P_{6-10}	6,3,2,7,8,9,10	**140**
P_{7-8}	7,8	17
P_{7-8}	7,2,3,6,5,8	**96**
P_{7-8}	7,2,3,6,9,8	**116**
P_{7-8}	7,2,1,3,6,5,8	**137**
P_{7-9}	**7,8,9**	66
P_{7-9}	**7,2,3,6,9**	**67**
P_{7-9}	7,8,5,6,9	72
P_{7-9}	7,2,1,3,6,9	**108**
P_{7-10}	**7,8,9,10**	**86**
P_{7-10}	**7,2,3,6,9,10**	**87**
P_{7-10}	7,8,5,6,9,10	92
P_{7-10}	7,2,1,3,6,9,10	**128**

P_{8-9}	8,9	49
P_{8-9}	8,5,6,9	55
P_{8-9}	8,7,2,3,6,9	**84**
P_{8-9}	8,7,2,1,3,6,9	**125**
P_{8-10}	8,9,10	69
P_{8-10}	8,5,6,9,10	75
P_{8-10}	8,7,2,3,6,9,10	**104**
P_{8-10}	8,7,2,1,3,6,9,10	**145**
P_{9-10}	9,10	20
P_{9-10}	-	-
P_{9-10}	-	-
P_{9-10}		

4-shortest paths and their path costs

Note: paths with excessive delay can be culled at this step

We now proceed to the next iteration. Of interest is the question of whether a deadlock situation could arise, resulting in cycling between a pair (or more) links having successive maximum utilisation. We will see that this can occur and a method will be proposed for breaking a cycle. The next few steps are described in the tables below.

Iteration 2

Link (2,7) now has the maximum utilisation.

Note that it is not necessary to re-compute k'th shortest paths. Paths with links (2,7) and (7,2) have their distances incremented by:

$$(\underset{u-v:P^{u-v}\in Q(2,7)}{Min} \Delta_{2,7}^{u-v})+1$$

Only some re-ordering of paths is necessary.

If link (3,6) had retained the maximum utilisation we would simply consider next $P^{3\text{-}8}$ and observe that $\Delta_{3,6}^{3-8}=5$

The new link cost for link (3,6) would be $m_{3,6}=12+\Delta_{\min}+1 \ =12+5+1=18$

This value is the old value -2

The new paths affected would be $P^{3\text{-}8}$ and $P^{4\text{-}8}$

We now proceed to identify Q(2,7)

Q(2,7) = $\{P^{u-v} : (2,7) \in P^{u-v}\}$

Path	Flow Demand
P^{1-7}	4 units
P^{1-8}	10 units
P^{2-7}	3 units
P^{2-8}	13 units
P^{3-7}	4 units
P^{4-7}	2 units
P^{6-7}	8 units

$u-v$	$d_{2,7}^{\text{u-v}}$	$\Delta_{2,7}^{\text{u-v}}$	f^{uv}	$P^{u-v}(d_{2,7}^{u-v})$
1-7	3	38	4	1,3,6,5,8,7
1-8	**2**	**4**	**10**	**1,3,6,5,8**
2-7	2	63	3	2,3,6,5,8,7
2-8	2	29	13	2,3,6,5,8
3-7	2	29	4	3,6,5,8,7
4-7	2	29	2	4,3,6,5,8,7
6-7	2	5	8	6,5,8,7

We see here the danger of reintroducing additional flow on link (3,6) -
in which case it is possible that we may either cycle or converge in a non-monotonic manner. That is, the objective function may increase in order to avoid a local optimum and then subsequently decrease. This would occur if transfer of flow to link (3,6) increased its utilisation, so that it once more had greatest utilisation, and then further cost increment to link (3,6) led to a greater reduction in its utilisation.

Continuing the principle of selecting the paths with
the least cost increment we would select $P^{1-8}(2)$ as the first path to deviate from using link (2,7) and effect this by incrementing the cost of link (2,7) by 4+1 = 5. That is, $m_{2,7} = 25 + 4 + 1 = 30$.

u-v	old path	new path	offered flow
1-8	1,**2,7**,8	1,**3,6**,5,8	10

↑ Paths containing link (2,7)

↑ Paths not containing link (2,7)

Link	Change in flow	New flow
(1,2)	-10	12
(2,7)	-10	34
(7,8)	-10	30
(1,3)	+10	44
(3,6)	+10	114
(6,5)	+10	64
(5,8)	+10	33

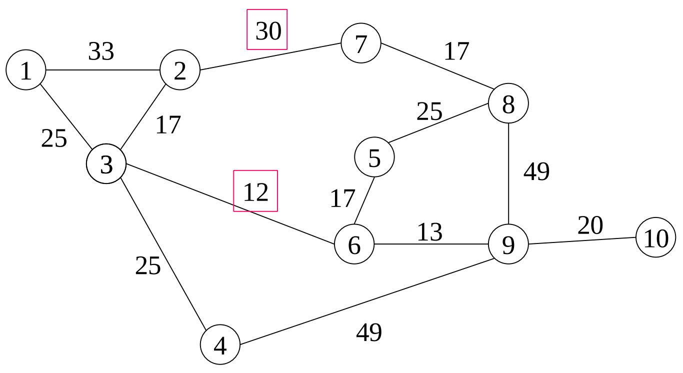
* similar flow values for links (2,1),(7,2),(8,7),(3,1),(6,3),(8,5)

Results after two iterations of the KSP cost adjustment heuristic

Link	Flow	Utilisation (decreasing)	Link	New flow	New utilisation
3,6	104	1,06122449	**(3,6)**	114	1.163265306
2,7	44	1,1	**(2,7)**	34	0.85
3,4	34	0,85	**(3,4)**	34	0.85
2,3	59	0,98333333	**(2,3)**	59	0.983333333
6,9	61	0,7625	**(6,9)**	61	0.7625
5,6	54	0,9	**(5,6)**	64	1.083333333
1,3	34	0,85	**(1,3)**	44	1,1
9,10	40	0,8	**(9,10)**	40	0.8
1,2	22	0,73333333	**(1,2)**	12	0.4
7,8	40	0,66666667	**(7,8)**	30	0.5
8,9	15	0,75	**(8,9)**	15	0.75
5,8	23	0,575	**(5,8)**	33	0.825
4,9	10	0.5	**(4,9)**	1 0	0.5

Average utilisation	0,77169676	Average utilisation	0.81287936
Maximum utilisation	1,1 (link (2,7)	Maximum utilisation	1,1 63265 (link (3,6)

Note: Both the maximum and average utilisation have increased after the second iteration but these values are still less than the original values.
Is this telling us that we didn't add enough cost to link (3,6) or that we will cycle?



New link costs

Continuing the next iteration we observe that the algorithm is repeatedly selecting links (3,6) and (2,7) for cost incrementation. We display the variable values for the next iteration and then address the matter of how we can avoid this repetition, or cycling.

1-8	$d_{3,6}^{1\text{-}8}$	$\Delta_{3,6}^{1\text{-}8}$	f^{u-v}	$P^{u-v}(d_{3,6}^{1\text{-}8})$
1-5	2	51	3	1,2,7,8,5
1-6	3	77	6	1,3,4,9,6
1-8	**2**	**1**	**10**	1,2,7,8
1-9	3	49	6	1,3,4,9
1-10	3	49	1	1,3,4,9,10
2-5	2	26	9	2,7,8,5
2-6	3	70	14	2,7,8,5,6
2-9	3	49	5	2,3,4,9
2-10	3	54	7	2,3,4,9,10
3-5	2	60	7	3,2,7,8,5
3-6	2	75	6	3,4,9,6
3-8	2	10	3	3,2,7,8
3-9	2	49	14	3,4,9
3-10	2	49	8	3,4,9,10
4-5	2	25	4	4,9,6,5
4-6	2	25	1	4,9,6
4-8	2	10	1	4,3,2,7,8

$$m_{3,6} = 12 + \Delta_{\min} + 1 = 12 + 5 + 1 = 18$$

Starting iteration 3

At iteration 2 we had:

u-v	old path	new path	offered flow
1-8	1,**2,7**,8	1,**3,6**,5,8	10

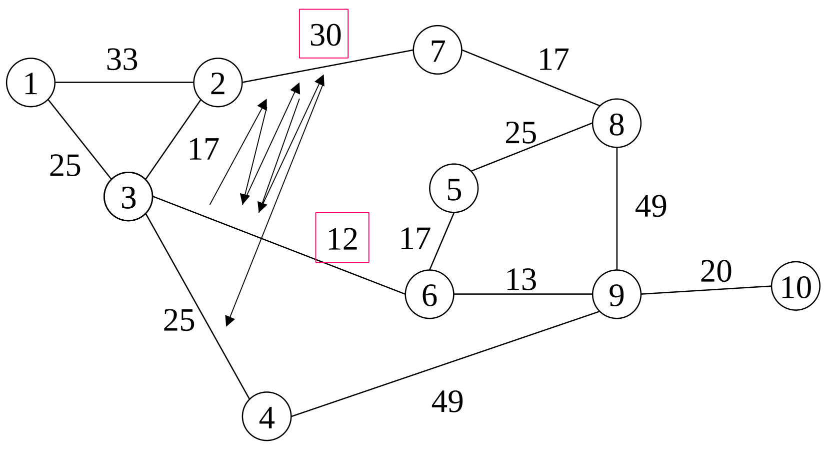
At iteration 3 we have:

u-v	old path	new path	offered flow
1-8	1,**3,6**,5,8	1,**2,7**,8	10

Path 1-8 is the path first affected by the increase in link distance

If we continue for a third iteration we will find that *(i*,j*)* is *(2,7)* and the old path is *1,3,6,5,8* and new path is *1,2,7,8* for O-D *1-8*.

That is, the algorithm would cycle. Each time the link distances are incremented for links (3,6) and (2,7) until their distances are large enough to cause re-direction of the traffic to other links – thus breaking the cycling



Cycling can occur between links (3,6) and (2,7) (i.e. the links with maximum utilisation – until their *costs* are sufficiently large relative to the other links and then the path deviation is to another path)

Link	**Flow**	**Utilisation**
3,6	104	1,06122449
2,7	44	1,1
3,4	34	0,85
2,3	59	0,98333333
6,9	61	0,7625
5,6	54	0,9
1,3	34	0,85
9,10	40	0,8
1,2	22	0,73333333
7,8	40	0,66666667
8,9	15	0,75
5,8	23	0,575
4,9	10	0.5

Average utilisation	0,77169676
Maximum utilisation	1,1 (link (2,7)

Note that we do not have to recompute the link utilisations the same paths have been involved in the cycle (see back 3 slides). The only effective change is that the link costs for link (3,6) and link (2,7) are being increased.

It would be possible to compute in advance the smallest increase in link costs for these two links to break the cycle from the original costs- thus avoiding more than one cycle.

Cycle-breaking

A cycle is detected by the following sequence of events:

(1) A link l_1 has max. utilisation. On checking for the smallest Δ for shortest paths using link l_1 we have a set of paths P_1 ,to which l_1 belongs, that lose flow and another set of paths P_2 ,to which l_1 does not belong, that receive flow.
We increment the link distance for l_1 by $\Delta + 1$.

(2) The next link with greatest utilisation is l_2.
The set of paths to which l_2 belongs that lose flow is P_2 , and the set of paths to which l_2 does not belong, that receive flow, is P_1.
(This implies that l_1 would once again be the link with greatest utilisation)

To break the cycle we seek the smallest $\Delta^{/}$ for shortest paths with link l_2 **not** involving l_1. This may involve 3rd shortest paths or 4th shortest etc.

Next we add the value of this new $\Delta^{/}$ to the distances of both links l_1 and l_2.

Will the strategy of always increasing the cost of the link with the **greatest utilisation** always work?
Referring to the following network we discuss a counter-example.

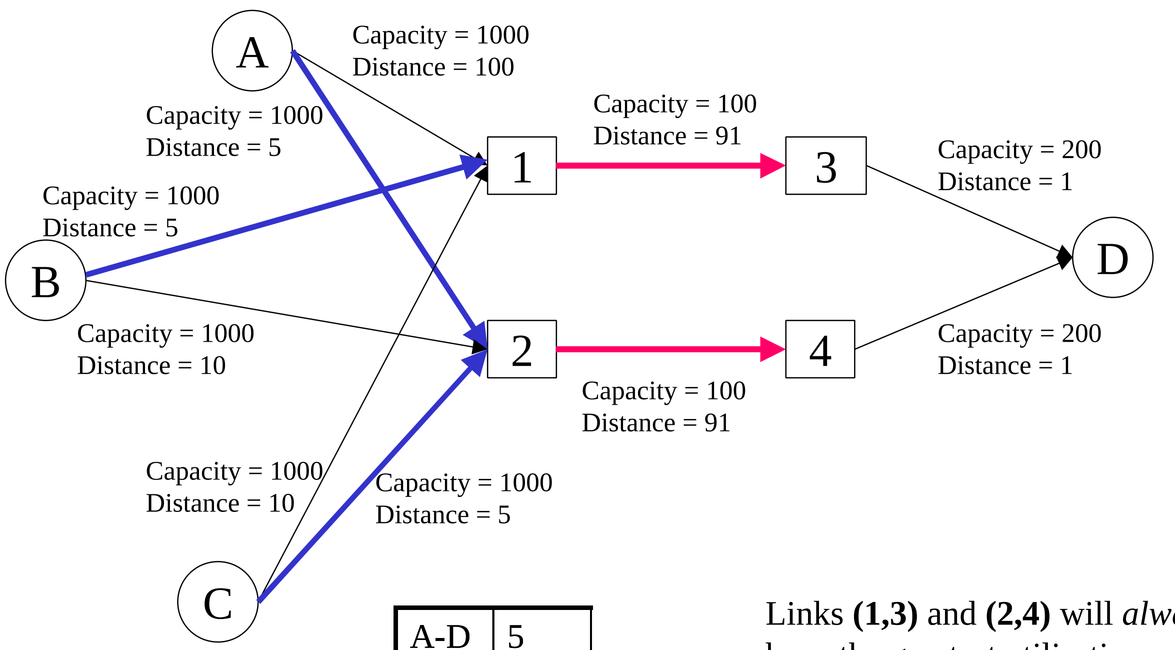
For the current example, suppose we detect a cycle at this stage:

1-8	$d_{3,6}^{1\text{-}8}$	$\Delta_{3,6}^{1\text{-}8}$	f^{u-v}	$P^{u-v}(d_{3,6}^{1\text{-}8})$
1-5	2	51	3	1,2,7,8,5
1-6	3	77	6	1,3,4,9,6
1-8	**2**	**1**	**10**	1,**2**,**7**,8
1-9	3	49	6	1,3,4,9
1-10	3	49	1	1,3,4,9,10
2-5	2	26	9	2,7,8,5
2-6	3	70	14	2,7,8,5,6
2-9	3	49	5	2,3,4,9
2-10	3	54	7	2,3,4,9,10
3-5	2	60	7	3,2,7,8,5
3-6	2	75	6	3,4,9,6
3-8	2	10	3	3,**2**,**7**,8
3-9	2	49	14	3,4,9
3-10	2	49	8	3,4,9,10
4-5	2	25	4	4,9,6,5
4-6	2	25	1	4,9,6
4-8	2	10	1	4,3,**2**,**7**,8

Smallest value of Δ

Second smallest Δ

The value of Δ chosen is 25 as $P^{u-v}(d_{3,6}^{1\text{-}8})$ does not include links (2,7) or (3,6)



A-D	5
B-D	40
C-D	45

Flow demands

Links **(1,3)** and **(2,4)** will *always* have the greatest utilisation

Shortest paths

$P_{A\text{-}D}$	A-2-4-D	97
$P_{A\text{-}D}$	A-1-3-D	192
$P_{B\text{-}D}$	B-1-3-D	97
$P_{B\text{-}D}$	B-2-4-D	102
$P_{C\text{-}D}$	C-2-4-D	97
$P_{C\text{-}D}$	C-1-3-D	102

Link utilisations

(A,1)	0
(A,2)	0.005
(B,1)	0.04
(B,2)	0
(C,1)	0
(C,2)	0.005
(1,3)	0.4
(2,4)	0.5
(3,D)	0.2
(4,D)	0.25

For this problem the optimal solution is to also have the B-D flow using link (1,3). Then both links with max. utilisation have values of **0.45. Can this be achieved?**

			Δ
$P_{A\text{-}D}$	A-**2-4**-D	97	95
$P_{A\text{-}D}$	A-1-3-D	192	
$P_{B\text{-}D}$	B-1-3-D	97	
$P_{B\text{-}D}$	B-2-4-D	102	
$P_{C\text{-}D}$	C-**2-4**-D	97	5
$P_{C\text{-}D}$	C-1-3-D	102	

We increase the distance of link (2,4) by 6

The effect of this is to change the shortest path for C-D to C-1-3-D.

A-D traffic will remain on path A-2-4-D. The new utilisations for the links are:

(A,1)	0
(A,2)	0.005
(B,1)	0.04
(B,2)	0
(C,1)	0.045
(C,2)	0
(1,3)	**0.85**
(2,4)	0.05
(3,D)	0.425
(4,D)	0.025

New shortest paths

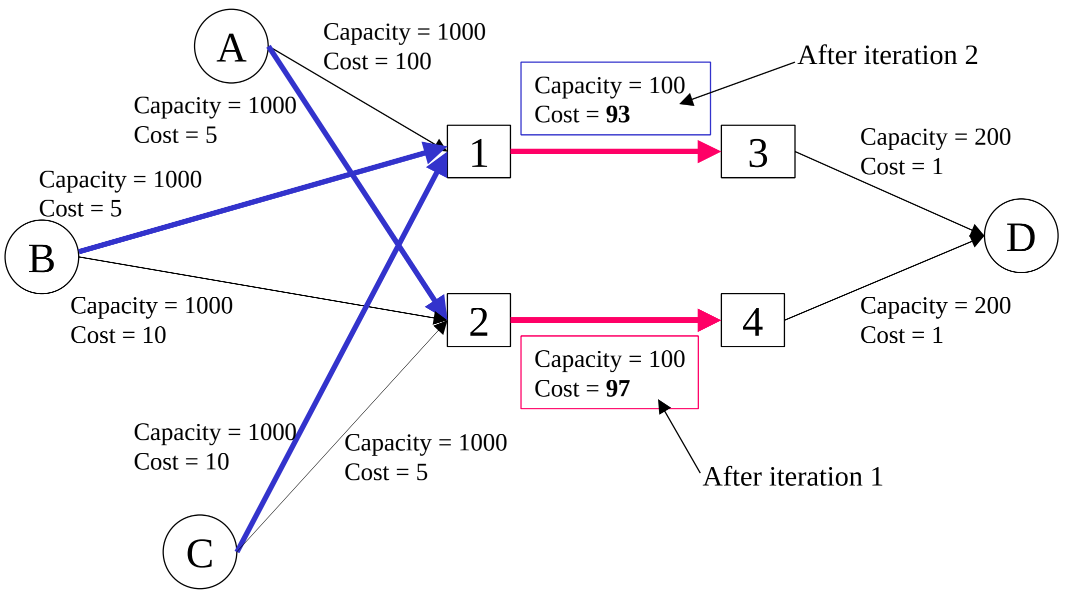
			Δ
$P_{A\text{-}D}$	A-2-4-D	97	
$P_{A\text{-}D}$	A-1-3-D	192	
$P_{B\text{-}D}$	B-**1**-**3**-D	97	5
$P_{B\text{-}D}$	B-2-4-D	102	
$P_{C\text{-}D}$	C-**1**-**3**-D	102	1
$P_{C\text{-}D}$	C-2-4-D	103	

We increase the distance of link (1,3) by 2.

The effect of this is to change the shortest path for C-D back to C-2-4-D. When we apply the cycle breaking technique we see that there are no k'th shortest paths that do not contain either link (1,3) or link (2,4).
So the algorithm would stop at this stage with the best solution:
Min (Max. utilisation) = 0.5
Thus missing the optimal solution.

We note that the method of selecting the initial link distances (i.e. *short distances for high capacity links*) could have annulled this counter-example. But it illustrates the potential for failure of the algorithm to find an optimal solution.

The link costs at this stage are as shown:



We next propose an extension to the algorithm to avoid the problem illustrated in the counter-example. The problem arose because of the algorithm's inability to recognize the potential for improvement from transferring flow to links with small utilisation. Thus, in addition to the procedure involving links with high utilisation, we now include the following procedure.

When a cycle results in termination, proceed as follows:
For the set of O-D pairs *u-v* that have been involved in the cycle, select a link with **minimum** utilisation and **decrease** its link cost by the smallest amount needed to create a **new** shortest path using that link. If a new shortest path cannot be found, apply the above for **all** O-D pairs.

u-v

Link distances

$P_{A\text{-}D}$	**A-2-4-D**	97
$P_{A\text{-}D}$	A-1-3-D	192
$P_{B\text{-}D}$	**B-1-3-D**	97
$P_{B\text{-}D}$	**B-2-4-D**	102
$P_{C\text{-}D}$	**C-1-3-D**	102
$P_{C\text{-}D}$	**C-2-4-D**	103

For this case link (A,1) would be selected and its distance reduced to 9. Hence path A-1-3-D can now be selected and the optimal solution found

(A,1)	0
(A,2)	0.005
(B,1)	0.04
(B,2)	0
(C,1)	0.045
(C,2)	0
(1,3)	**0.85**
(2,4)	0.05
(3,D)	0.425

Link utilisations

For our example, application of this rule results in re-allocation of the flow between A and D, and the optimal solution is then found, namely a min-max utilisation of 0.45.
During the implementation of the algorithm, we work with integer link costs. When the first set of link costs is generated there is no guarantee that unique shortest paths

exist for all source-destination pairs. The rule for updating link costs ensures that paths that are re-ordered have distinct costs but not all shortest paths may be re-ordered during the implementation of the algorithm. At the conclusion of the algorithm, we need to ensure that the set of link costs produce unique paths for each source-destination pair. This is an essential requirement of the IP routing protocol. The uniqueness can be guaranteed by means of a simple perturbation of the final link costs (this perturbation will not alter the optimal set of paths). The approach adopted is similar to that of [2].
Suppose that the network has *p* links, numbered 1,2,3…p and let σ(p) be a permutation of the integers 1,2,3…p. The cost of link *j*, w_j, is replaced by $w_j + 2^{\sigma(j)-p-1}$. It can be seen that the sum of the perturbation terms is less than 1 since each perturbation is selected from the terms of the geometric series 1/2,1/4,1/8,... Alternatively, this perturbation can be made after the initial set of integer link costs have been assigned at the first step of the algorithm.

Comment on an Inversion of the Shortest Path Problem

Given a network and a set of single paths between each source-destination pair satisfying the shortest path principles one can find a set of link costs that would correspond to these shortest paths. In [11] this inverse shortest path problem is solved using Linear Programming. These link costs are **not** unique. In general, there are many solutions to the problem that do not just involve a constant multiplier. For example, if a randomly selected set of symmetric link costs are generated such that $w_{ij} = w_{ji}$ and shortest paths determined, a set of link costs such that $w_{ij} \neq w_{ji}$ may be found as a valid solution.

3.2 Summary

The proposed heuristic should be tested thoroughly against known results from application of the method of [11]. It has the following potential advantages:

- Large network problems may now be considered
- Computational time will be reduced (compared to mathematical programming methods)
- It will be easy to evaluate alternative solutions near the optimal solution.

In the next section we examine the issues related to the use of decomposition methods, for the IP routing problem, with the view to extending the range of applicability of the method proposed in [11].

4. Network Decomposition

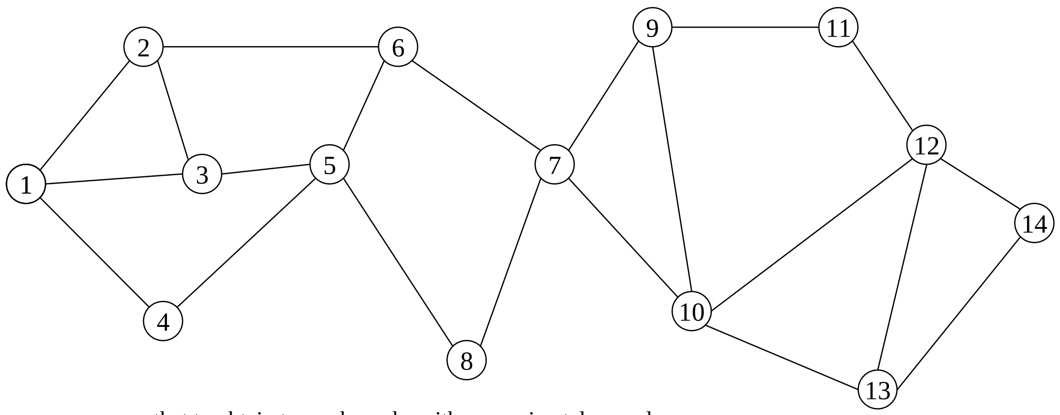
Suppose that we are given a large undirected network G(V,E), where V denotes the set of nodes or vertices of the network and E is its set of edges or links. We can identify two main objectives.
First, we wish to find an efficient algorithm to *automatically* decompose the network into disjoint connected subgraphs having the following desirable properties:

- Each subgraph is to have approximately the same number of nodes
- The number of separator nodes (see below) used in the decomposition is minimized.

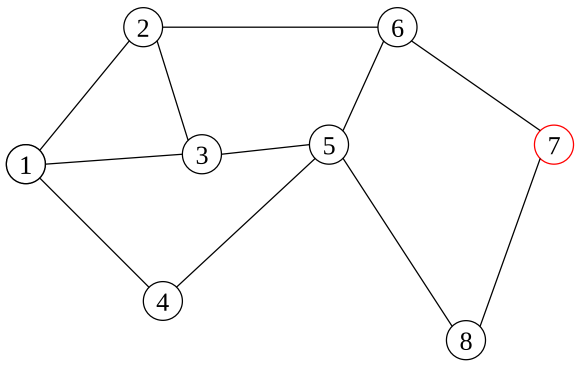
Secondly, we seek an efficient algorithm for independently applying the IP network routing optimization algorithm to each subgraph and then combining the results to obtain the optimal (or near-optimal) solution for the original network G.

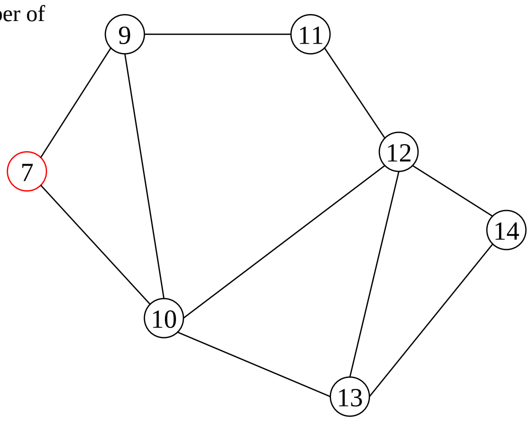
4.1 First objective

We see that the following network can be subdivided into two subgraphs satisfying our two desirable properties by separating at node 7. We seek an automatic method for finding such a node separator set, from the input link data that describes the network topology.



By eye we can see that to obtain two subgraphs with approximately equal numbers of nodes (i.e. *balanced* subgraphs) and with a minimal number of *node separators*, we select node 7 as a *separator set S.*





The general approach we adopt for our first objective is to start with any network partition [*S,B,W*] where *S* is a *node (vertex) separator set*. *B* and *W* are the two disjoint subgraphs induced by *S*. Assume that *B* is the bigger of the two subgraphs.
We consider strategies to *reduce the imbalance and to reduce the separator size* based on the Dulmage-Mendelsohn decomposition [4] and rely significantly on ideas from [1].
First we require some terminology and definitions.

The *adjacent set* of a vertex *v* is given by $Adj(v) = \{u \neq v \mid (u,v) \in E\}$.
A *vertex separator S* is a subset of *V* if the subgraph induced by the vertices in *V* but not in *S* has more than one connected component (similar definition for *edge separator*). A separator is *minimal* if no subset of it forms a separator.
An *Evaluation function* is proposed in [1]: $g[S;B;W] = |S|(1 + a\,\dfrac{\max\{|B|,|W|\}}{\min\{|B|,|W|\}})$
The parameter α determines the relative significance of minimizing the number of nodes in the separator set compared to balancing the sizes of the two subgraphs.
We wish to minimise this function.
The *interior of a set Y* is $Int(Y) = \{y \in Y \mid Adj(y) \subseteq Y\}$
That is, the *adjacent nodes of Y* are all in Y
The *boundary of the set Y* is the set of nodes **not** in Y that are adjacent to Y. It is denoted by $Adj(Y)$.
$Adj(Y) = \{v \in V \setminus Y \mid (y,v) \in E \text{ for some } y \in Y\} = (\bigcup_{y \in Y} Adj(y)) \setminus Y$
The *border of Y* is the boundary of the interior of Y (i.e. a subset of Y).

If $[S,B,W]$ is a two set partition of G and Z is a subset of S then
$Z \rightarrow W$ denotes the *move* of the subset Z from S to W.
This creates the new partition:

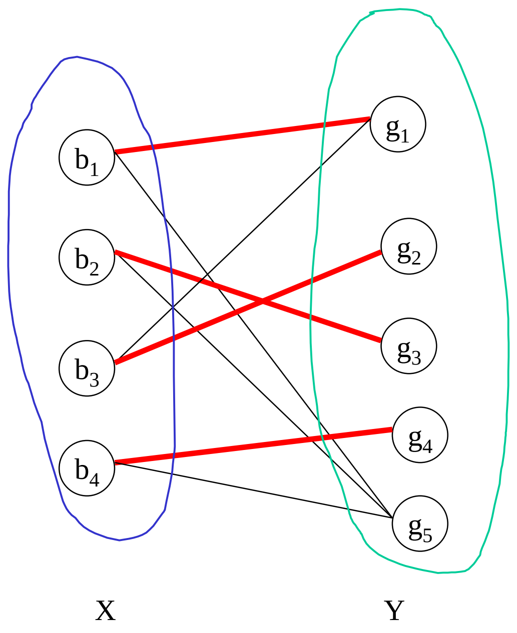
$$B_{Z \rightarrow W} = B \setminus Adj(Z),\ \ W_{Z \rightarrow W} = W \cup Z,\ \text{and}\ \ S_{Z \rightarrow W} = (S \setminus Z) \cup (Adj(Z) \cap B)$$

The new separator set is *S* minus Z plus nodes from B in Adj (Z)
Thus the new separator size is $|S_{Z \rightarrow W}| = |S| - |Z| + |Adj(Z) \cap B|$.
Therefore if we can find a subset Z of *S* such that $|Z| > |Adj(Z) \cap B$,
then the move of Z to *W* will result in a *reduction of the separator set size.*
The question now is: "How do we find an optimal set Z that achieves our purpose?"
The solution comes from the theory of bipartite graph matching.
Some further definitions are needed.

A *bipartite graph* is an undirected graph whose set of nodes can be divided into two disjoint sets (X and Y) such that every edge (link) has one endpoint in each set.

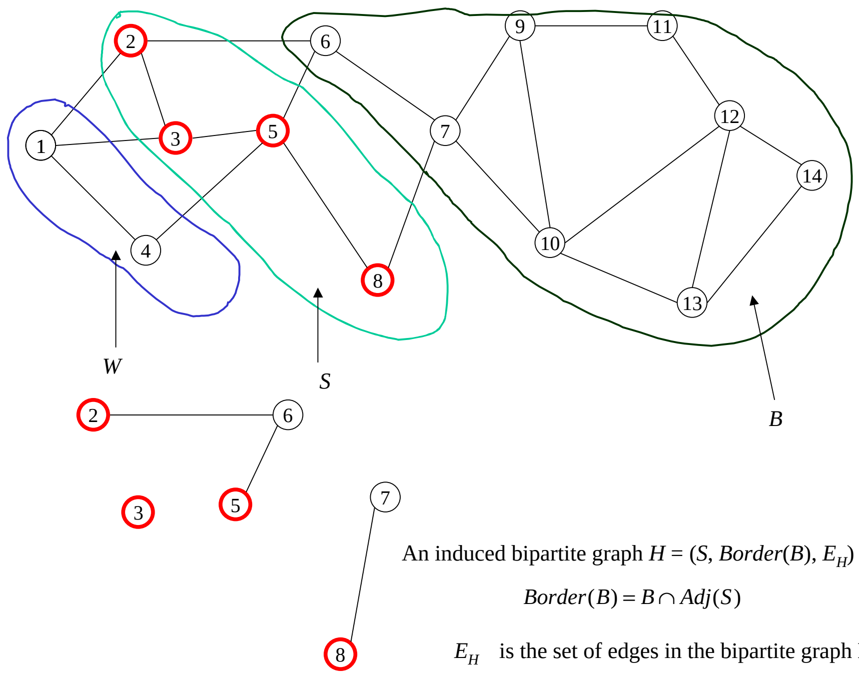
A *matching* of a bipartite graph *H* is a subset *M* of edges such that no two edges in this subset have a node in common.
A node that is incident to an edge in *M* is said to be *covered*, otherwise it is *exposed*.
The number of edges in M is said to be the *size* of the matching.
A *maximum matching* is one of largest size for the bipartite graph.
A *complete matching* from X to Y in a bipartite graph is a one-to-one correspondence between the vertices in X and a *subset* of the vertices in Y, with the property that the corresponding vertices are joined.
If (x,y) belongs to the matching *M* then x = mate(y) and y = mate(x).

The concept of a complete matching is readily explained with the classical marriage problem.
Given a finite set of boys each of whom knows several girls, under what conditions can we marry off the boys in such a way that each boy marries a girl *he knows*? A matching is illustrated below with the edges in the matching being given as thick lines.



Example of a **complete matching** from the nodes in X to the nodes in Y

For the example network, with separator set {2,3,5,8}, the sets *W* and *B* are as shown in the following figure.
The bipartite graph induced by the separator set is also shown.

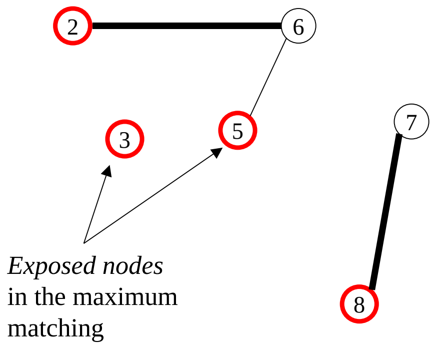


An induced bipartite graph $H = (S, Border(B), E_H)$

$Border(B) = B \cap Adj(S)$

E_H is the set of edges in the bipartite graph H

There are two maximal mappings from the nodes in S to the nodes in $B \cap Adj(S)$, one of which is shown below by the thick lines. We see that the matching is not complete (nodes 3 and 5 are not paired with nodes in $B \cap Adj(S)$).



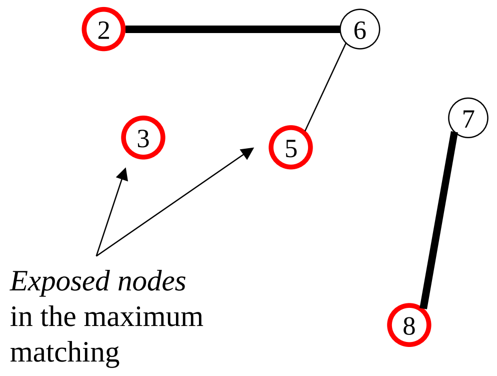
A *maximum* matching M
(this is **not** a *complete* matching)

We want to find a subset Z of S such that $|Z| > |Adj(Z) \cap B|$. To answer this question we make use of Hall's theorem [5].

Hall's Theorem (1935)

The bipartite graph H has a *complete matching* of
S into B if and only if for every subset Z of S, $|Z| \leq |Adj(Z) \cap B|$.

Alternative statement:
A necessary and sufficient condition for a solution of the marriage problem is that every set of k boys collectively knows at least k girls.
.
We want to find a subset Z of S such that $|Z| > |Adj_H(Z) \cap B|$.
Thus Hall's theorem can be used to provide a necessary and sufficient condition for the existence of a size-improving subset Z of S, *i.e. that H does not have a complete matching from S into B.*
For our example, we do not have a complete matching; therefore we can find a size-improving subset of S.
To find the size-improving subset, we require the concept of an *alternating path*:
Given a matching M, a simple path (sequence of unrepeated nodes)
$x_1, x_2, \dots, x_k$ is an *alternating path* if alternate edges belong to the matching.



For this example, their are two exposed nodes in S. Therefore we can find a size-improving subset Z of S.

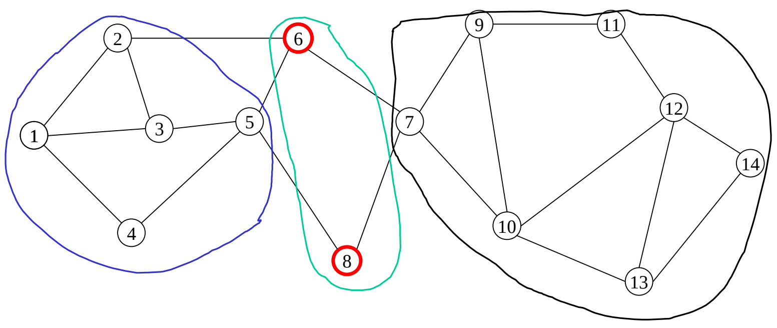
A theorem due to Liu [7] provides information on the magnitude of the size improvement.

Liu's Theorem (1989)

Let $x \in S$ be an exposed node in a maximum matching of H.
Define $S_x = \{s \in S \,|\, \text{s is reachable from } x \text{ via alternating paths}\}$.
Then $|S_x| - |Adj_H(S_x)| = 1$

The alternating path 2,6,5 in the maximum matching allows us to now place the nodes 2 and 5 in *W* and to put the node 6 into *S*. The isolated node 3 can also be placed in *W*.

At this stage, we have the new decomposition of the network as shown below.



We note that the set S was not a minimal vertex separator (nodes 2,5 and 8 would have given a separation of the graph into two subgraphs). Node 3 did not feature in the matching.
We now have a better-balanced decomposition of the network and a reduction in the number of nodes in the vertex separator.
This process can be made even more efficient.

In applying Liu's theorem, the set S_x can be determined by performing an *alternating breadth- first search* starting from the exposed node x.
The first improvement is to use *all* exposed nodes in S to find a subset Z in S that maximises the decrease in separator size.
A theorem due to Pothen and Fan [10] is useful in helping to select the optimal set X from S.

Pothen and Fan's Theorem (1990)

Define a set of nodes $S_I = \{s \in S \mid s \text{ is reachable from some exposed node in } S \text{ via alternating paths}\}$

Then:

$$|S_I| - |Adj_H(S_I)| \; > \; 0$$

$$|S_I| - |Adj_H(S_I)| = \operatorname*{Max}_{Z \subseteq S}\{|Z| - |Adj_H(Z)|\}$$

S_I is the *smallest* subset of S with this maximum value $|S_I| - |Adj_H(S_I)|$

Although the subset S_I gives the maximum reduction in separator size, one might prefer a smaller reduction in exchange for a better balance between the two subgraphs.
If X_0 is any subset of the exposed nodes in S then $Z = \bigcup \{S_x \mid x \in X_0\}$ satisfies the condition: $|Z| \; > \; |Adj_H(Z) \cap B|$.

We have seen how to improve the decomposition for the case that the maximum matching in the induced bi-partite graph is not complete. When the matching is complete the above theorems fail to improve the decomposition, In this case we can proceed by applying a result based on the Dulmage-Mendelsohn Decomposition [3].

The Dulmage-Mendelsohn Decomposition

(Originally published in the Canad. J. Math. In 1958
Re-stated in our context by Pothen & Fan in 1990)

Suppose that the induced bipartite graph from a given partition [*S,B,W*] is *H(S,B)* and that we have a maximum matching *M* on *H*.
The Dulmage-Mendelsohn *decomposition of the vertex separator set S* is the decomposition of S into 3 disjoint subsets: $S = S_I \cup S_X \cup S_R$.

$S_I = \{s \in S \mid$ s is reachable from some exposed node in S via alternating paths$\}$

$S_X = \{s \in S \mid$ s is reachable from some exposed node in B via alternating paths$\}$

$S_R = S \setminus (S_I \cup S_X)$

The set S_R is the set of **r**emaining nodes in *S*, after S_I and S_X have been determined.

Theorem:

The Dulmage-Mendelsohn decomposition S_I, S_X, S_R is *independent of the maximum matching* used to define the alternating paths

Theorem (due to Pothen & Fan):

The set $S_I \cup S_R$ satisfies:

$|S_I \cup S_R| - |Adj_H(S_I \cup S_R)| = |S_I| - |Adj_H(S_I)|$

$S_I \cup S_R$ is the largest subset of S with the maximum value

$\underset{Z \subseteq S}{Max}\{|Z| - |Adj_H(Z)|\}$

S_I is the smallest subset of *S* that gives the maximum reduction in separator size
$S_I \cup S_R$ is the largest subset of *S* that gives the maximum reduction in separator size

Thus, moving either of S_I or $S_I \cup S_R$ will achieve the same size reduction but the balance for the resulting partition may be better for one or the other of the two moves.

By symmetry, we can also decompose *Border (B)* into 3 disjoint node sets B_I, B_X, B_R . For example:
$B_X = \{b \in B \mid$ b is reachable from some exposed node in S via alternating paths$\}$.

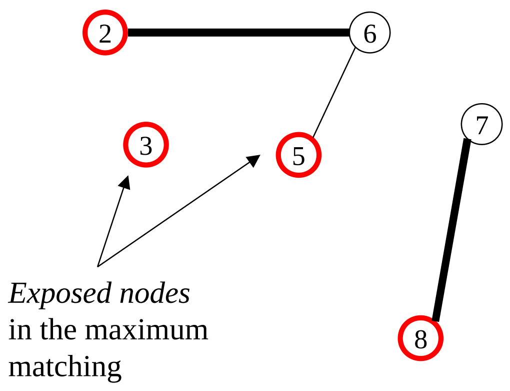
Theorem:
$S_X = Adj_H(B_I)$ and $B_X = Adj_H(S_I)$

S_X can be given by the adjacent set of B_I, the set of reachable nodes in *B* from internal exposed nodes via alternating paths.
B_I can be found by forming the alternating breadth-first forest from the set of exposed nodes in *B*.

We illustrate the above results as follows:

Adjacency Theorem:

$S_X = Adj_H(B_I)$	S_X can be given by the adjacent set of B_I, the set of reachable nodes in *B* from internal exposed nodes via alternating paths
$B_X = Adj_H(S_I)$	B_I can be found by forming the alternating breadth-first forest from the set of exposed nodes in *B*



$S_I = \{5,2\}$
$S_X = \{\}$
$S_R = \{8\}$
$B_I = \{\}$
$B_X = \{6\}$
$B_R = \{5\}$

$\mathrm{Adj}_H(B_I) = \{\}$

$\mathrm{Adj}_H(S_I) = \{6\}$

Comments on the Stopping Condition

If a separator-improving subset of *S* can be found satisfying $|Z| > |Adj_H(Z) \cap B|$ then we can attempt to use it to gain a better balance between the subgraph sizes. Otherwise if no such subset can be found, no reduction in separator size by graph matching is possible. This occurs when the maximum matching is complete. The current separator *S* is already of minimum size among covering separator subsets of $S \cup Border(B)$.

We may still be able to improve the imbalance ratio: $\dfrac{\max\{|B|,|W|\}}{\min\{|B|,|W|\}}$

We can search for a subset *Z* in *S* with $|Adj_H(Z)| = |Z|$.

A move of *Z* to the smaller subgraph *W* will replace *Z* by $|Adj_H(Z)|$ in *S*. There will be no change in separator size, but there may be a reduction in the imbalance.

If S_I is empty, size reduction is not possible for the vertex separator set, but the subset S_R can be used to reduce the imbalance.

The following lemma is used to show that in the case that S_I is empty (and it is then not possible to reduce the size of the vertex separator set), $\mathbf{S_R}$ is the key to finding a **balance-improving** separator subset.

Lemma:

Let $S_I = \{\}$ and consider a subset Z of S.

If $Z \cap S_X$ is non-empty then $|Z| < |Adj_H(Z)|$.

Theorem:

Let $S_I = \{\}$. The separator subset S_R is the largest subset of S such that its size is the same as the size of its adjacent set.

We list two of the properties of S_I:

$|S_R| = |B_R|$

$$Adj_H(S_I \cup S_R) = B_X \cup B_R$$

Thus if S_I is empty then by definition B_X is empty and therefore

$$Adj_H(S_R) = B_R$$

Therefore, when the separator subset S_I is empty, the move of S_R to W will give a new separator:

$$S_{S_R \to W} = (S \cup B_R) \setminus S_R$$

so that

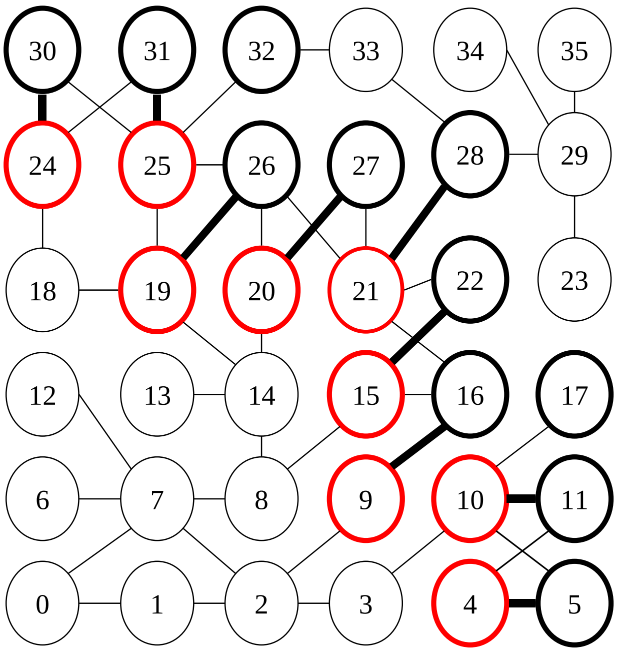
$$| S_{S_R \to W} | = | (S \cup B_R) \setminus S_R | = | S |$$

Theorem:

Let [S,B,W] be a partition with $|B| > |W|$ and $S_I = \{\}$.
Given a subset Z with $|Z| = |\mathrm{Adj}_H(Z)|$ the move of Z to W
Will reduce the evaluation function if and only if $|Z| < |B| - |W|$.

Example

Dulmage-Mendelsohn Decomposition:



$S_I = \{\}$

$S_X = \{4,10,24,25\}$

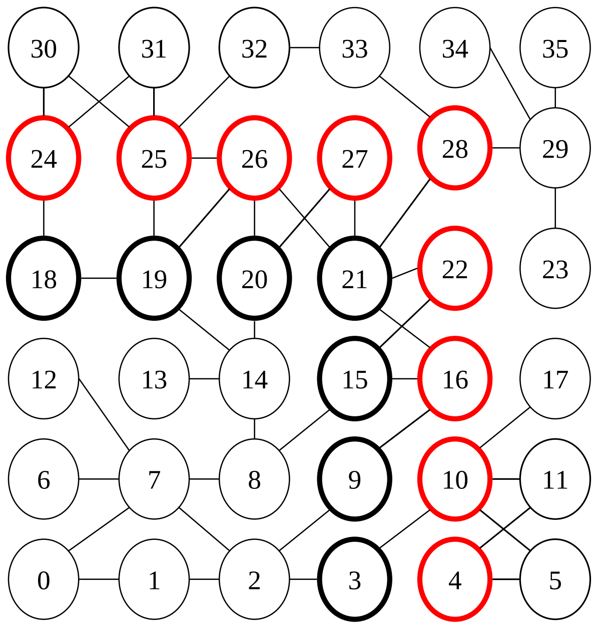
$S_R = \{9,15,19,20,21\}$

$B_I = \{5,11,17,30,31,32\}$

$B_X = \{\}$

$B_R = \{16,22,26,27,28\}$

There is a *complete matching* of *S* into *B*, hence $S_I = \{\}$
The size of the separator set cannot be reduced.
Moving S_R from *S* to *W* and B_R to *S* gives a new separator set with the same size and also gives better balance of the subgraphs



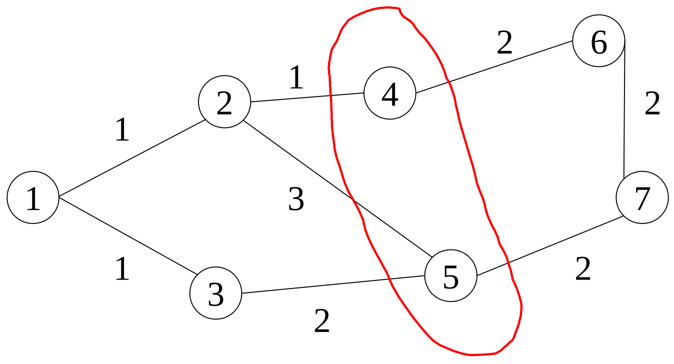
Shifting S_R = {9,15,19,20,21} from *S* to *W* and B_R = {16,22,26,27,28} from *B* to *S*

4.2 Second objective

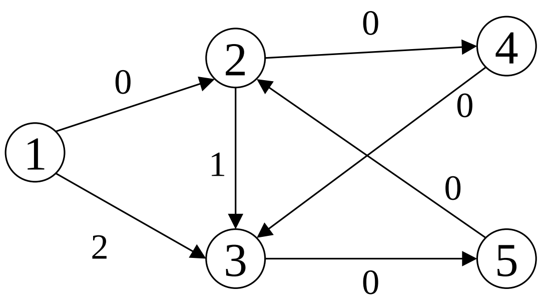
We now seek to find an efficient algorithm for independently applying the IP network link cost optimization algorithm [11] to each subgraph and then combining the results to obtain the optimal link costsfor the **original network G.**

Preliminary Comments

It is not possible simply to apply the algorithm to the two subgraphs separately and to use the link costs obtained for the entire network. Although a shortest path must consist of shortest subpaths (the optimality principle), the following example illustrates that the concatenation of two shortest paths may not give a shortest path.
Let $S_{u\text{-}v}$ denote the *shortest* (cheapest) path from *u* to *v*.
$S_{1\text{-}4}$ = 1-2-4 (of length 2)and $S_{4\text{-}7}$ = 4-6-7 (of length 4). The concatenation of these two shortest paths gives the path 1-2-4-6-7 of length 6, but $S_{1\text{-}7}$ = 1-3-5-7 is of length 5.
Suppose *u* is in *W* and *v* is in B.
We need to consider the shortest distance from node *u* to all nodes in the separator set and then the shortest distance from such nodes *s* to *v*.



It is pointed out in [8] that the optimality principle does not apply to k'th shortest paths:



Let $P_{u\text{-}v}(k)$ denote the k'th shortest path between u and v.

$P_{1\text{-}2}(1)$ = 1,2 with distance 0

$P_{1\text{-}2}(2)$ = **1,3**,5,2 with distance 2

$P_{1\text{-}3}(1)$ = 1,2,4,3 with distance 0

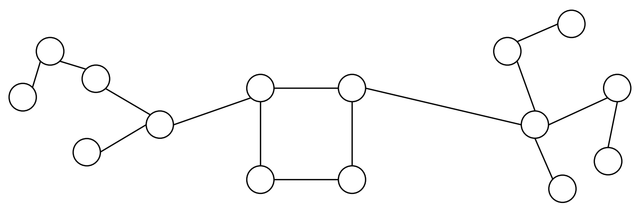
$P_{1\text{-}3}(2)$ = 1,2,3 with distance 1

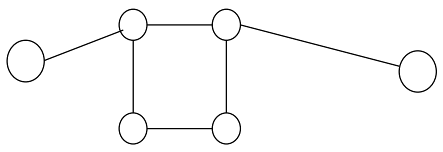
$P_{1\text{-}3}(3)$ = **1,3** with distance 2

The 3rd shortest path from 1 to 3 is a subpath of the 2nd shortest path from 1 to 2

We see that a *k*'th shortest path may exist containing a *j*'th shortest subpath with *j* > *k*.

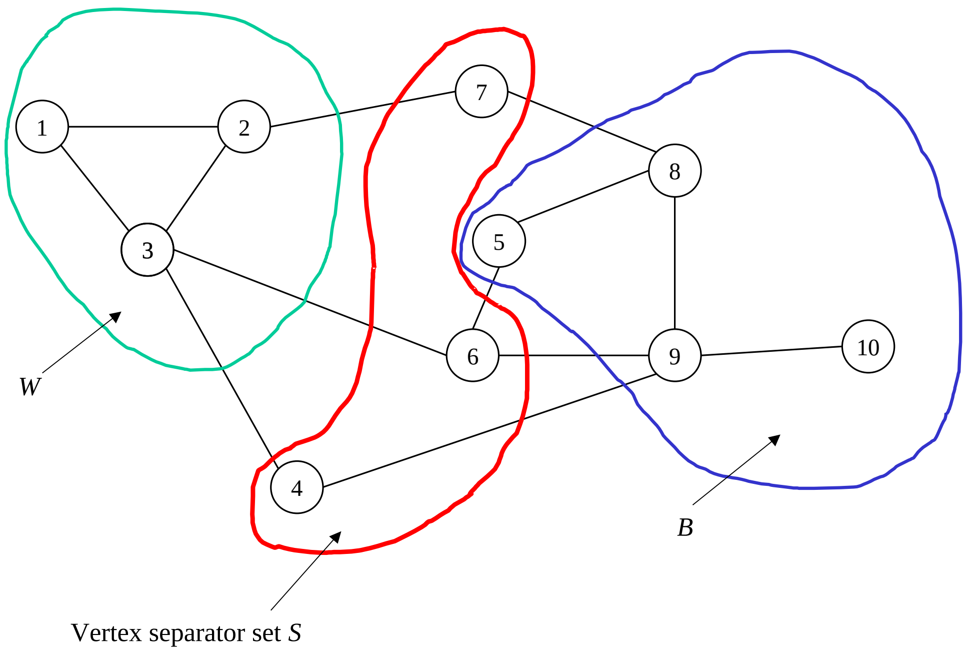
If the network has identifiable subgraphs comprising tree structures, there is an immediate simplification possible. Since a tree has a unique path between any two member nodes, the costs given to the links of the tree may be arbitrary. For example, the network below may be reduced to a mesh-type network and two leaves.



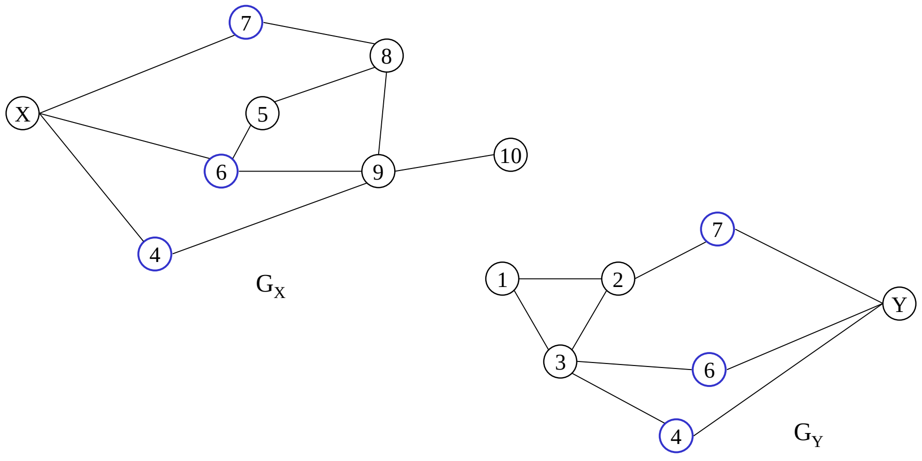


A Preliminary Approach to the Problem

Consider the following decomposition of a network.



We form from the previous network two associated networks by adding dummy nodes X and Y.



Note that the structures of *B* and *S* are retained in the network G_X, with dummy node X, whilst the sub-graph W is condensed to a single node. Similarly, the structure of *W* and *S* are retained in the network G_Y, with dummy node Y, whereas B is condensed to the single node Y.
Suppose that the input data for the network is as follows.

	1	2	3	4	5	6	7	8	9	10
1	0	8	5	12	3	7	4	10	6	1
2	8	0	2	11	9	14	3	13	5	7
3	5	2	0	3	7	6	4	3	14	8
4	12	11	3	0	4	1	2	1	7	3
5	3	9	7	4	0	3	7	2	9	5
6	7	14	6	1	3	0	8	10	4	2
7	4	3	4	2	7	8	0	7	2	1
8	10	13	3	1	2	10	7	0	3	9
9	6	5	14	7	9	4	2	3	0	4
10	1	7	8	3	5	2	1	9	4	0

Actual Flow demands between O-D pairs

	1	2	3	4	5	6	7	8	9	10
1	0	30	40							
2	30	0	60				40			
3	40	60	0	40		98				
4			40	0					20	
5					0	60		40		
6			98		60	0			80	
7		40					0	60		
8					40		60	0	20	
9				20		80		20	0	50
10									50	0

Original Link capacities

The input data for the two related subgraphs G_X and G_Y are obtained by the following rules.

The demand from node X (or Y) to a node *s* in *S* is the sum of the demands to *s* from the nodes aggregated together to form node X (or Y).

Let *s* be a node in *S*. The capacity of link (*X*,*s*) is $\sum_{(k,s)\in L; k\in W} c_{ks}$
That is, the capacity of link (X,*s*) is the sum of the capacities of links adjacent to s in *W*.
Let *s* be a node in *S*. The capacity of link (*Y*,*s*) is $\sum_{(k,s)\in L; k\in B} c_{ks}$
That is, the sum of the capacities of links adjacent to s in *B*.

Applying the above rules, the flow demands for Gx and G_Y are:

	X			4	5	6	7	8	9	10
X	**0**			**26**	**19**	**27**	**11**	**26**	**25**	**16**
4	**26**			0	4	1	2	1	7	3
5	**19**			4	0	3	7	2	9	5
6	**27**			1	3	0	8	10	4	2
7	**11**			2	7	8	0	7	2	1
8	**26**			1	2	10	7	0	3	9
9	**25**			7	9	4	2	3	0	4
10	**16**			3	5	2	1	9	4	0

Flow demands for G_X

	1	2	3	4	5	6	7	8	9	Y
1	0	8	5	12		7	4			20
2	8	0	2	11		14	3			34
3	5	2	0	3		6	4			32
4	12	11	3	0		1	2			15
5										
6	7	14	6	1		0	8			19
7	4	3	4	2		8	0			17
8										
9										
Y	20	34	32	15		19	17			0

Flow demands for G_Y

	X	2	3	4	5	6	7	8	9	10
X	0			40		98	40			
2										
3										
4	40			0					20	
5					0	60		40		
6	98				60	0			80	
7	40						0	60		
8					40		60	0	20	
9				20		80		20	0	50
10									50	0

Capacities for G_X

	1	2	3	4	5	6	7	8	9	Y
1	0	30	40							
2	30	0	60				40			
3	40	60	0	40		98				
4			40	0						20
5										
6			98			0				140
7		40					0			60
8										
9										
Y				20		140	60			

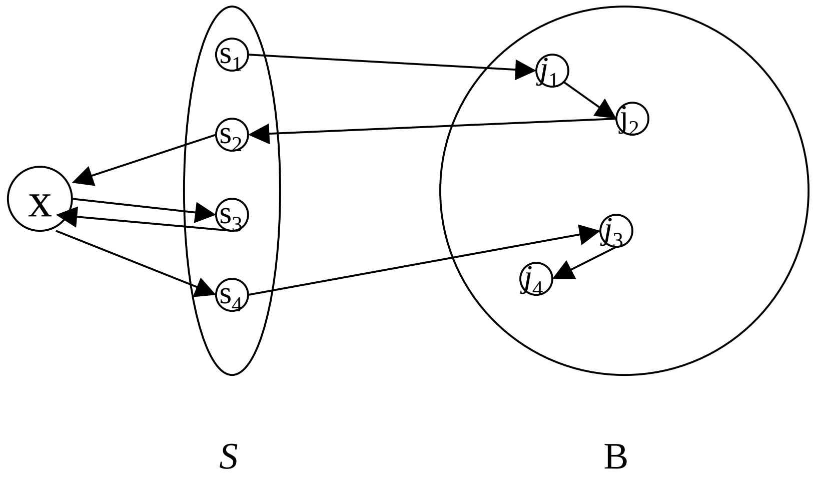
Capacities for G_Y

We next use the LP to find the optimal routing in the two networks G_X and G_Y such that we minimize the maximum link utilisation (or minimize an objective function also involving the average utilisation over the network).

The optimal solution to the LP is a set of paths between each pair of nodes, satisfying the shortest (cheapest) path principle.
We record for each source *u* and destination *v* the optimal paths between them in G_X and G_Y, respectively. If u (v) $\in$ W (B) in G_X (G_Y) the path is to X (or Y as the case may be). Let us denote these optimal paths by $P_X(u\text{-}v)$ and $P_Y(u\text{-}v)$ respectively.
Note that at this stage it is not necessary to know the link costs that would normally be generated by the usual application of the algorithm in [11].
For the example under consideration the optimal paths $P_X(u\text{-}v)$ and $P_Y(u\text{-}v)$ are shown in the table on the following page These were found by solving for optimal paths on G_X and G_Y separately..
Looking at the first entry, the source node is node 1 and the destination is node 10. In G_Y, node 10 $\in$ Y and $P_Y(1\text{-}10)$ is 1-3-6-Y. The path between nodes 1 and 10 in G_X is X-6-9-10. Note that the nodes 1,3 belong to X and the nodes 9,10 belong to Y. The node 6 in *S* is common to both paths in G_Y and G_X. This is an example of what we will call *compatible paths*.

Definition: Two paths, $P_Y(u\text{-}v)$ and $P_X(u\text{-}v)$ are said to be *compatible* if they consist of path segments that belong entirely to W, B and *S* in matching sequences with nodes in *S* having an identical sequence for the two paths $P_Y(u\text{-}v)$ and $P_X(u\text{-}v)$.

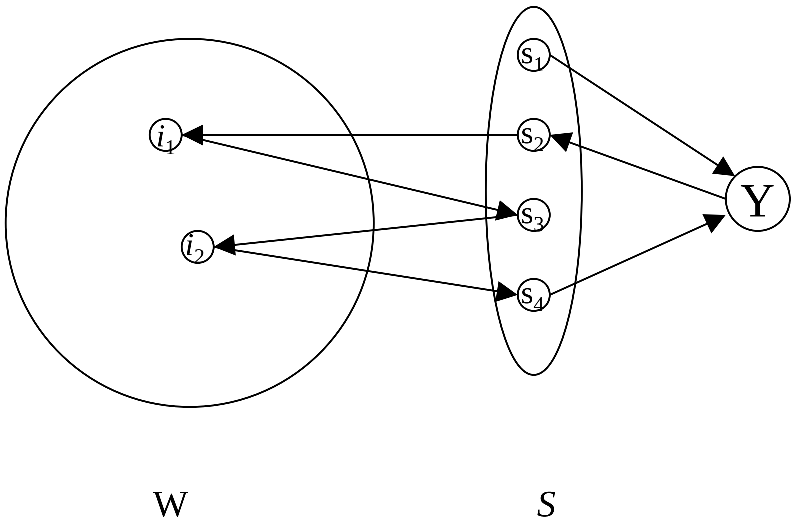
The paths 1-3-**6**-Y and X-**6**-9-10 each have 3 segments belonging in sequence to X- *S* - Y and they have the common node 6 in *S*. The concept is illustrated with a further example.



The path $P_X(u\text{-}v)$ with $u = s_1$ and $v = j_4$ is : s_1–j_1-j_2-s_2-X-s_3-X-s_4-j_3-j_4. Its segments belong to *S*-B-*S*-X-*S*-X-*S*-B respectively.

u	**v**	**s(u-Y)**	**s(X-v)**	**Path in Gy**	**Path in Gx**	**Path in G**
1	10	6	6	1-3-6-Y	X-6-9-10	1-3-6-9-10
1	9	6	6	1-3-6-Y	X-6-9	1-3-6-9
1	8	6	7	1-3-6-Y	X-7-8	
1	7	7	7	1-2-7	X-7	1-2-7
1	6	6	6	1-3-6	X-6	1-3-6
1	5	6	6	1-3-6-Y	X-6-5	1-3-6-5
1	4	4	4	1-3-4	X-4	1-3-4
1	3		both in X	1-3		1-3
1	2		both in X	1-2		1-2
2	10	6	6	2-3-6-Y	X-6-9-10	2-3-6-9-10
2	9	6	6	2-3-6-Y	X-6-9	2-3-6-9
2	8	6	7	2-3-6-Y	X-7-8	
2	7	7	7	2-7	X-7	2-7
2	6	6	6	2-3-6	X-6	2-3-6
2	5	6	6	2-3-6-Y	X-6-5	2-3-6-5
2	4	4	4	2-3-4	X-4	2-3-4
2	3		both in X	2-3		2-3
3	10	6	6	3-6-Y	X-6-9-10	3-6-9-10
3	9	6	6	3-6-Y	X-6-9	3-6-9
3	8	6	7	3-6-Y	X-7-8	
3	7	6&7	7	3-6-Y-7	X-7	
3	6	6	6	3-6	X-6	3-6
3	5	6	6	3-6-Y	X-6-5	3-6-5
3	4	4	4	3-4	X-4	3-4
4	10	4	4	4-Y	4-9-10	4-9-10
4	9	4	4	4-Y	4-9	4-9
4	8	4	4	4-Y	4-9-8	4-9-8
4	7	both in S	both in S	4-Y-7	4-9-8-7	4-9-8-7
4	6	both in *S*	both in S	4-Y-6	4-X-6	
4	5	4	4 & 6	4-Y	4-X-6-5	
5	10	both in Y	6		5-6-9-10	5-6-9-10
5	9	both in Y	6		5-6-9	5-6-9
5	8	both in Y			5-8	5-8
5	7	Y to *S*	7	Y-7	5-8-7	5-8-7
5	6	Y to *S*	6	Y-6	5-6	5-6
6	10	6	6	6-Y	6-9-10	6-9-10
6	9	6	6	6-Y	6-9	6-9
6	8	6	6	6-Y	6-5-8	6-5-8
6	7	both in *S*	6&7	6-Y-7	6-5-8-7	6-5-8-7
7	10	7	7	7-Y	7-8-9-10	7-8-9-10
7	9	7	7	7-Y	7-8-9	7-8-9
7	8	7	7	7-Y	7-8	7-8
8	10	both in Y			8-9-10	8-9-10
8	9	both in Y			8-9	8-9
9	10	both in Y			9-10	9-10

Now consider the path $P_Y(u\text{-}v)$: s_1,Y,s_2,,i_1,s_3,i_2,s_4,Y.



Its segments belong to S-Y-*S*-W-*S*-W-*S*-Y respectively. Now condensing W to X and *B* to Y we see that both paths become *S*-Y-*S*-X-*S*-X-*S*-Y. In addition, the transit nodes in *S* are visited in the same sequence ($s_!$, s_2, s_3, s_4) and with identical members in each of the sets *S*. Thus the paths are compatible.
We can define and form the *expansive concatenation* of the two paths, denoted by $P_X(u\text{-}v) \otimes P_Y(u\text{-}v)$, to obtain: $s_1–j_1\text{-}j_2\text{-}s_2\text{-}\ i_1\text{-}s_3\text{-}\ i_2\text{-}s_4\text{-}j_3\text{-}j_4$.
This is obtained by replacing X in $P_X(u\text{-}v)$ by its corresponding elements in W from $P_Y(u\text{-}v)$. Since $P_X(u\text{-}v) \otimes P_Y(u\text{-}v) = P_Y(u\text{-}v) \otimes P_X(u\text{-}v)$, we could also replace Y in $P_Y(u\text{-}v)$ by its corresponding elements in B from $P_X(u\text{-}v)$.
We note that the expansive concatenation is, in this case, a connected simple (loopless) path.

Returning now to the table above, all simple and connected paths in G formed by expansive concatenation are listed in the last column under the heading "Path in G". It is seen that 39 of the 45 source-destination pairs have compatible paths in G_X and G_Y respectively. In all cases the expansive concatenations of their paths are simple connected paths. We remind the reader that our data matrices are symmetric and paths from *v* to *u* are identical with paths from *u* to *v* in any feasible solution.
This example provides an instance where the analysis of G_X and G_Y separately does not lead directly to an optimal solution to the original (larger) network G[V,A]. In some cases, the optimal solution can be found directly from the optimizations on G_X and G_Y . The following theorem can be used to identify such cases.

Theorem

If for all source-destination pairs *u* and *v* in G[V,A] the expansive concatenation $P_X(u\text{-}v) \otimes P_Y(u\text{-}v)$ is a simple connected path then these paths give the optimal set of paths in G[V,A].

Proof: The optimal flow allocation on G_X minimizes the maximum utilisation over all possible path selections on *B and* S (satisfying the shortest path principles). Similarly, the optimal flow allocation on G_Y minimizes the maximum utilisation over all possible paths in *W* and *S.* Such utilisations in G are not altered if the paths are compatible and the expansive concatenations are simple connected paths. The paths are also feasible in G (satisfying shortest path principles). Thus it is not possible to improve on this path set.

A consequence of the above theorem is that if for all source-destination pairs *u* and *v* in G[V,A] the expansive concatenation
$P_X(u\text{-}v) \otimes P_Y(u\text{-}v)$ is a simple connected path then the sets of link costs obtained (for links of G) by applying the algorithm directly to the two smaller graphs G_X and G_Y may be used as the optimal link costs for the original network.

We still need to consider the case when some of the paths $P_X(u\text{-}v)$ and $P_Y(u\text{-}v)$ are not compatible and the question: "Can flows be re-allocated to new paths, retaining shortest path principles, such that an optimal (or near optimal) solution can be found for the original network G?"

Lower Bound for the solution on G(V,E)

The link utilisations obtained for the links in G(V,E), from the optimal solutions to the two smaller graphs G_X and G_Y , provide a lower bound for the min-max solution.

Approaches for the case $P_X(u\text{-}v)$ and $P_Y(u\text{-}v)$ are not compatible

There are a number of different ways that one could proceed. Although it is not guaranteed that an optimal solution can be obtained, without some type of exhaustive search, a near optimal solution is sought.

Method 1
(1) Compute the link costs from the solutions for the networks G_X and G_Y as in [11].
(2) If the costs for links in *S* differ for G_X and G_Y, randomly select the cost for each link in *S* from either its cost in G_X or G_Y.
(3) Use these costs as the initial link costs for the KSP Cost Adjustment Heuristic.

If we examine the table summarizing the results for our example and look at the 6 cases where paths are not compatible we can identify the causes of incompatibility.

For source-destination pairs 1-8, 2-8 and 3-8, *u* is in W and *v* in B. The incompatibility is due to different transit nodes being selected for paths in G_X and G_Y.
For source- destination pair 3-7, *u* is in W and *v* in *S*. The incompatibility is due to one solution being a path from W to *S* and the other solution being a path for W to a different node in *S* and then to Y and back to *S*.
In the case of source-destination pair 4-6, both nodes are in *S*. The optimal path in G_X is via X (that is, the subgraph W) but the optimal path in G_Y is via Y (B).
For source-destination pair 4-5, *u* is in *S* and *v* in B. There is a conflict between selection of a path *u*-Y versus a path *u*-X-*S*-Y.

Method 2
(1) Order the source-destination pairs with incompatible paths in decreasing order of their offered traffic loads.
(2) Select in turn each ordered pair of nodes and compute feasible paths from *u* to the destination *v* conforming with the (compressed) path solution from G_X and satisfying the shortest path principles. Select the path most favourable to the objective (e.g. minimizing the maximum link utilisation). Repeat with the path solution from G_Y. Select from these two paths the one most favourable to the objective.
(3) Compute as in [11] the link costs.

Comments on steps (2) and (3)
Because of pre-processing possibilities for the second LP described in [11], the computation of the link costs can be done directly on the original network G[V,A].
The selection of a path for a source-destination pair with incompatible paths involves both removal of its load from the links of G in G_X (or G_Y) and addition of the load to the selected path.
The set of paths obtained from the expansive concatenation of compatible paths satisfy shortest path principles. Thus, if a source-destination pair *k-v* has been assigned a path then if the path from node *u* includes node *k* then it must have a common path segment from *k* to *v*.
It is ,of course, also possible after completion of step (3) to use these costs as the initial link costs for the KSP Cost Adjustment Heuristic.

5. Conclusions
We have considered the problem of computing optimal sets of paths between source-destination pairs in large IP networks according to the model proposed in [11]. The traffic demands between the pairs of nodes, the link capacities and the topological structure of the network are given. The problem is to allocate the traffic streams to routes such that either the maximum link utilisation is minimized, or the average link utilisation of the network is minimized, or the objective is a combination of these two targets. To be feasible, the set of paths selected, for all pairs, must be constrained

to satisfy shortest path principles.
Two approaches were developed that are potentially fast and able to handle large networks. The first, based on use of k'th shortest paths, iteratively modifies link costs to achieve the selected objective. The second method seeks to apply network decomposition effectively such that the LP technique of Staehle, Köhler and Kohlhaas [11] may be implemented on each subgraph in an iterative algorithm. It remains now to evaluate the approaches by applying the methods to a significant number of test networks.

References

[1] C. Ashcraft, J.W.H. Liu, "Applications of the Dulmage-Mendelsohn Decomposition and Network Flow to Graph Bisection Improvement", SIAM J. Matrix Anal. Appl., Vol. 19, No. 2, pp. 325-354, April 1998.

[2] A. Bley, M. Grotschel, R. Wessaly, "Design of Broadband Virtual Private Networks: Model and Heuristics for the B-Win, Pre-print, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1998.

[3] E. Dijkstra, "A Note on Two Problems in Connection with Graphs", Numerical Mathematics, Vol. 1, pp. 395-412, 1959.

[4] A. Dulmage, N. Mendelsohn, "Coverings of Bipartite Graphs", Canad. J. Math. , Vol. 10, pp.517-534, 1958.

[5] P. Hall, "On Representatives of Subsets", J. London Math. Soc., Vol. 10, pp.26-30, 1935.

[6 N. Katoh, T. Ibaraki, H. Mine, "An Efficient Algorithm for k Shortest Simple Paths", Networks, Vol. 12, pp.411-424, 1982.

[7] J.W.H. Liu, "A graph Partitioning Algorithm by Node Separators", ACM Trans. Math. Software, Vol. 15, pp.198-219, 1989.

[8] E.Q.V. Martins, M.M.B. Pascoal, J.L.E. Santos, "The K Shortest Loopless Paths Problem", Research Note, Dept. Mathematics, Uni. Coimbra, Portugal, July 1998.

[9] E.Q.V. Martins, M.M.B. Pascoal, J.L.E. Santos, "A New

Improvement for a K Shortest Paths Algorithm", Research Report, Dept. Mathematics, Uni. Coimbra, Portugal, Nov. 1999.

[10] A. Pothen, C. Fan, "Computing the Block Triangular Form of a Sparse Matrix"; ACM Trans. Math. Software, Vol. 16, pp. 303-324, 1990.

[11] D. Staehle, S. Köhler, U. Kohlhaas, "Towards an Optimization of the Routing Parameters for IP Networks", Univ. Würzburg Institute of Computer Science Research Report, 2000.

[12] J.Y. Yen, "Finding the K Shortest Loopless Paths in a Network", Management Science Vol. 17, pp.712-716, 1971.

Appendix- *K* Shortest Paths in a Network

The algorithms available in the present literature fall into two main categories, those that can be applied to find *k*'th shortest *simple* paths (those that are without loops) and those that find all *k*'th shortest paths. The second class of algorithm is not directly suitable for finding simple paths. For our problem we must generate *k*'th shortest simple paths. The main algorithms for generating *k*'th shortest simple paths are:

- **Yen's algorithm** (1971).
  Applicable to directed and undirected networks
- Katoh, Ibaraki & Mine's Algorithm (1982).
  Applicable to directed networks
- Martins, Queiros & Pascoal's Algorithm (1999).
  Applicable to directed and undirected networks

All of the above algorithms use a *path deviation* approach. Labelling type algorithms are not suitable owing to the fact that the "optimality principle" for shortest paths does not hold for *k*'th shortest paths.
The main characteristics of the path deviation algorithms are:
•The *k*'th shortest path p_k is the shortest path taken from a candidate set X.
• To form the set X, we start with the *k*-1'th shortest path and consider in turn each node on the path except the destination node. The considered node is called the deviation node and a *new path* is formed for each deviation node, *i*. The new path is loopless and one that is not in $\{p_1, p_2, ..., p_{k-1}\}$

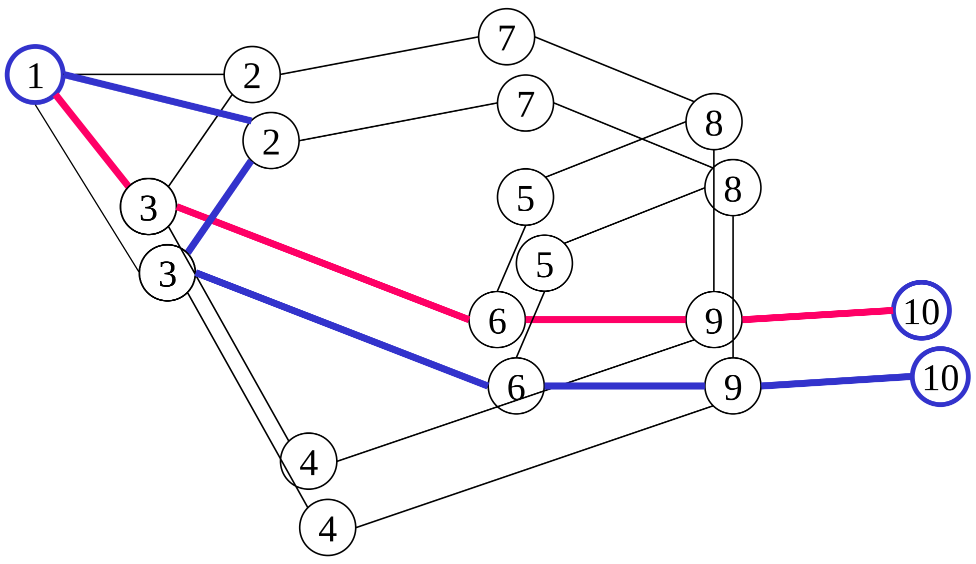
•The *new path* is formed from the concatenation of p_{si} the subpath of p_{k-1} from s to i and p^*_{it} where p^*_{it} is a shortest path from i to the destination node t satisfying the condition that the new concatenated path is loopless and one that is not in $\{p_1, p_2, ..., p_{k-1}\}$

••The concatenation is denoted by $p_{si} \oplus p^*_{it}$

Many algorithms exist for the k'th shortest unconstrained path problem (loops allowed), but these are not suitable for our purpose.

Yen's algorithm is explained with an example.

The k'th Shortest Path Pseudo-trees



A **pseudo-tree** composed of shortest and second shortest paths. Note that all nodes of the original graph may be repeated except for the source.

Yen's Algorithm

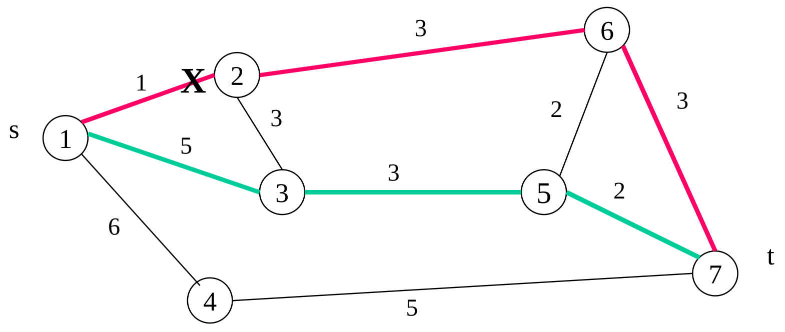
Let p_k denote the k'th shortest path, from node s to node t, just selected from X. Let p_{sv} be the subpath of pk from node s to node v (the deviation node).

To form p^*_{vt} we

•Remove from the graph all nodes of p_{sv} , except node v, (this effectively also removes the links between these nodes). This is to ensure that no cycles are found in the new path

•Remove links (v,w) that belong to the pseudo-tree $\mathcal{T}_k$. That is, remove links (v,w) that have belong to the previously found shortest paths $\{p_1, p_2, ..., p_{k-1}, p_k\}$
•Find the shortest path, p^*_{vt} , in the remaining network, from v to t and then the concatenation $p_{sv} \oplus p^*_{vt}$ is formed and the path entered into the set X.
•We repeat this process for all deviation nodes on p_k and the shortest path in X becomes p_{k+1}.

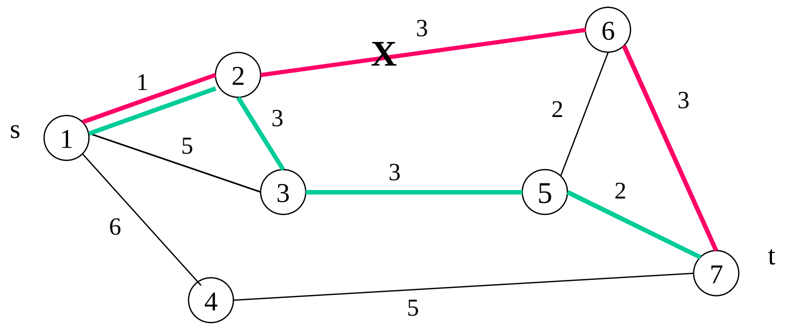
Example



P_1 = 1,2,6,7

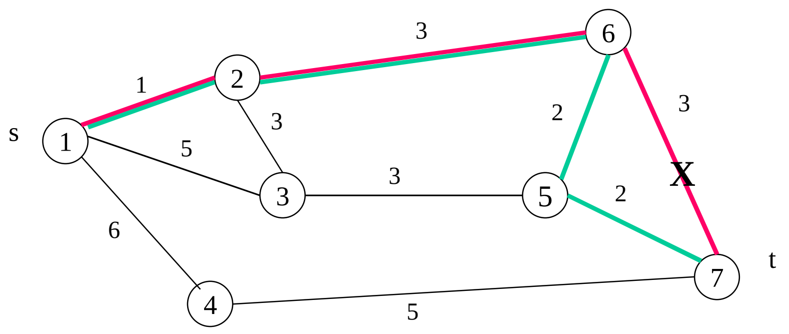
Deviation node *v*	**Path p_{sv}**	**Path p^*_{vt}**	$p_{sv} \oplus p^*_{vt}$
1	-	1,3,5,7	1,3,5,7
2			
6			

Example (cont.)



Deviation node *v*	**Path p_{sv}**	**Path p^*_{vt}**	$p_{sv} \oplus p^*_{vt}$
1	-	1,3,5,7	1,3,5,7 (10)
2	1,2	2,3,5,7	1,2,3,5,7 (9)
6			

Example (cont.)



Deviation node *v*	**Path p_{sv}**	**Path p^*_{vt}**	$p_{sv} \oplus p^*_{vt}$
1	-	1,3,5,7	1,3,5,7 (10)
2	1,2	2,3,5,7	1,2,3,5,7 (9)
6	1,2,6	6,5,7	**1,2,6,5,7 (8)**

P_2 = 1,2,3,5,7 (length =8)

Continuing the example, the *k*'th shortest paths are :

P_1	1,2,6,7	7
P_2	1,2,6,5,7	8
P_3	1,2,3,5,7	9
P_4	1,3,5,7	10
P_5	1,4,7	11
P_6	1,2,3,5,6,7	12
P_7	1,3,5,6,7	13
P_8	1,3,2,6,7	14
P_9	1,3,2,6,5,7	15

Although efficient *k*'th shortest simple and non-simple path algorithms exist for both directed and undirected networks, these are designed for the case of a single origin and a single destination. In our problem we may, of course, apply an algorithm repeatedly to each source-destination pair in the network. This, however, may not be the most efficient way to compute *all pairs k*'th shortest paths. In discussion with Martins [8,9], it seems that no efficient algorithm is known for the all pairs *k*'th shortest path problem. This remains an open research area. Currently, the method adopted in [11] is based on an exhaustive search (with the number of possible paths reduced by imposing hop-limits) and this is followed by a sorting process.