# Some Tools for Finding Deadlock-free Routings in Computer Networks Based on Linear Programming and Extensions

Ralph Lorentzen
ralphlo@simula.no

Tecnical Report 2016-03

May 23, 2016

**Abstract**

We consider an uncapacitated directed network where the nodes are switches or CPU-s and the arcs are channels going from a port on one switch/CPU to a port on another switch/CPU. The ports are assumed to have sufficient bandwidth capacity to accommodate any traffic that one may choose to route through them. There may be more than one channel in each direction connecting two ports, and each physical channel may have subchannels called virtual channels. We describe several approaches based on linear programming and some other approachesfor finding deadlock-free short routes for a given set of required connections between pairs of CPU-s. Some routes may have been specified beforehand. We also consider extension of the models to reflect limits on the maximal use of a single channel. Then we describe an approach to reroute requirements when a single channel fails. Finally we look into the problem of coupling together two or more networks where deadlock-free routings exist for each subnetwork.

Keywords: directed networks, deadlocks, linear programming

## 1 Introduction

We consider an uncapacitated directed network where the nodes are switches or CPU-s with many ports and the arcs are channels going from a port on one switch/CPU to a port on another switch/CPU, see Figure 1. The ports are assumed to have sufficient bandwidth capacity to accommodate any traffic that one may choose to route through them. There may be more than one channel in each direction connecting two ports, and each physical channel may have several subchannels called *virtual channels*. Within a switch we are free to connect any port to any other port.

A CPU may be connected to more than one switch, and more than one CPU may be connected to one switch. In this network we have connection requirements, each from an origin CPU to a destination CPU. We want to route

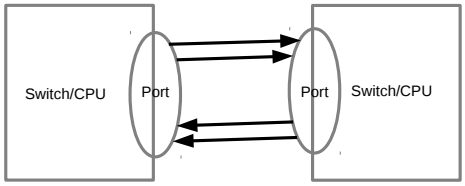


Figure 1: Channels with two virtual channels connecting two ports

these requirements through a set of short directed routes so that the routing is deadlock-free. The bandwidth of each requirement is specified to be one bandwidth unit.

In order to conveniently model the connection requirements we augment the network by *artificial ports* and *artificial virtual channels*. For each port in a CPU we establish a corresponding artificial port. Between the (real) port and its corresponding artificial port we establish an artificial channel with one virtual channel in each direction, see Figure 2.

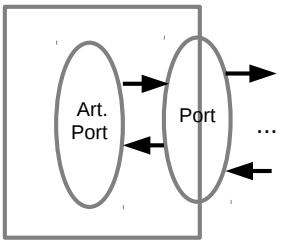


Figure 2: Artificial port and corresponding artificial virtual channels in a CPU

In treating this problem it is useful to operate with a different network, the *virtual channel/dependency network*. A pair of consecutive virtual channels $(l^s, l^e)$ is called a *dependency*, see Figure 3. The virtual channel/dependency network is the directed network $(C, D)$ where the nodes are virtual channels and the arcs are dependencies.

We have experimented with three different formulations of the problem where we have had success with the last one only. Nevertheless, we shall for the sake of completeness describe all three models.

For general networks the problem of finding the shortest deadlock-free paths is NP-complete (see [1] where it is shown that deciding whether a deadlock-free routing exists or not is NP-complete). For bidirectional networks we know, however, that deadlock-free routings exist, but it is still difficult to find optimal ones. Therefore we do not expect to find guaranteed optimal solutions for other than very small (and uninteresting) problems, and we are currently satisfied with solutions we can find using heuristics. We can, however, get an idea of the quality of the solutions we find by comparing them with the solutions we get when we just find shortest paths ignoring deadlock avoidance.

## 2 Model 1 – Notation

We introduce the following additional notation:

| | |
|---|---|
| $M_r$ | set of artificial virtual channels where requirement $r$ can originate |
| $N_r$ | set of artificial virtual channels where requirement $r$ can terminate |
| $D_h^s$ | set of dependencies that start in switch $h$ |
| $D_h^e$ | set of dependencies that end in switch $h$ |
| $l_d^s$ | the start virtual channel node for dependence arc $d$ |
| $l_d^e$ | the end virtual channel node for dependence arc $d$ |
| $t$ | the number of requirements |
| $c$ | the number of requirements plus the number of channels in the network |

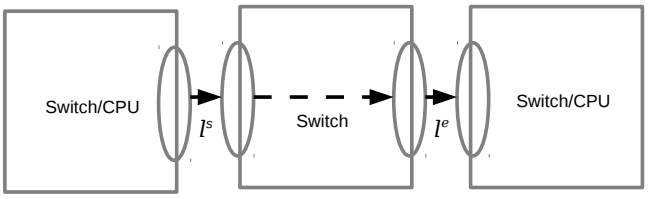


Figure 3: Dependency $(l^s, l^e)$

# 3 Model 1 – Formulation

The deadlock-free routing problem is formulated as a mixed integer programming problem. We introduce the following variables:

| | |
|---|---|
| $y_d$ | total flow in dependency $d$, unrestricted |
| $x_l$ | deadlock avoidance number associated with virtual channel node $l$, unrestricted |
| $f_{dr}$ | flow associated with requirement $r$ in dependency $d$, $\geq 0$ |
| $\delta_d$ | 0 if dependency $d$ is used, 1 otherwise |
| $z$ | total use of dependencies |

$f_{dr}$ is fixed for requirements $r$ with preset routing. $\delta_d$ is not generated for preset dependencies.

The variables $x_l$ are introduced in order to model deadlock avoidance. A dependence cannot be used unless its end virtual channel node has an $x$-value larger than the $x$-value of its start virtual channel node.

The problem becomes:

$$\text{minimize } z = \sum_d y_d \tag{3.1}$$

subject to

$$-y_d + \sum_r f_{dr} = 0 \quad \text{for all dependencies } d \tag{3.2}$$

$$y_d + t\delta_d \leq t \quad \text{for all not preset dependencies } d \tag{3.3}$$

$$y_d - c\delta_d - x_{l_d^e} + x_{l_d^s} \leq 0 \quad \text{for all dependencies } d \tag{3.4}$$

$$\sum_{l_d^s \in M_r} f_{dr} = 1 \quad \text{for all requirements that are not preset} \tag{3.5}$$

$$\sum_{l_d^e \in N_r} f_{dr} = 1 \quad \text{for all requirements that are not preset} \tag{3.6}$$

$$\sum_{d \in D; l_d^s = l} f_{dr} - \sum_{d \in D; l_d^e = l} f_{dr} = 0 \quad \text{for all } l \text{ and all } r \text{ without preset routing} \tag{3.7}$$

(3.2) defines the total use of a dependency. (3.3) and (3.4) secure that no deadlock occur. (The $x$-s cannot increase around a cycle of dependencies.) (3.5) and (3.6) secure that the requirements are satisfied. (3.7) secures that the input is equal to the output for each virtual node (balance equations).

# 4 Model 1 – Solution method

We have used the open source linear programming package GLPK. Using GLPK's integer programming option is hopeless since we cannot take advantage of the characteristics of the problem. Therefore we have written our own depth-first search approach where we solve a sequence of linear programming (LP) relaxations. The root node in the search tree corresponds to the solution of the relaxed LP where all $\delta_d$ -s are allowed to be any number between 0 and 1. The next node to be selected in the search tree is based on finding a maximum $y_d$ (i.e. a most used dependence) with fractional $\delta_d$. We round this $\delta_d$ to zero and solve the corresponding relaxed problem. If the problem is feasible, we proceed to a next maximal $y_d$ with fractional $\delta_d$ and round the $\delta_d$ to zero. We continue with this until we arrive at a situation where rounding to zero gives an infeasible problem. Then we round the most recently zero-rounded $\delta_d$ to one instead. If this gives a feasible solution, we continue rounding a new $\delta_d$ to zero. If not, we step up the search tree unfixing $\delta_d$ till we reach the most recent problem where a $\delta_d$ was rounded to zero. We round this $\delta_d$ to one instead and continue from there.

We cannot hope to find a solution which minimizes $z$, but by stopping at the first feasible solution with integer $\delta_d$-s the hope was to find a reasonably good solution. For small toy problems (e.g. for a two-dimensional torus with 9 switches, 9 CPU-s and 72 connection requirements) this works fine. However, for larger problems GLPK gets into numerical trouble after 700-800 roundings. So we would need a more robust and powerful LP package if we want to explore this approach further. We have also analyzed the branch-and-bound process and found that the minimization of the sum of the $y$-s implies that the same loops and infeasibilities are formed again and again. This gave us the motivation for searching for an alternative approach.

**Assigning virtual channel numbers in acyclic networks** Later we shall need a quick way of assigning numbers to the virtual channels in an acyclic virtual channel/dependency network such that the channel number of the end virtual channel of a dependency has a higher number than the start virtual channel of the dependency. We describe the approach here since it uses the same notation that was used in Model 1. We easily achieve such numbers by solving the following LP problem:

$$\text{minimize } z = \sum_l x_l \tag{4.1}$$

subject to

$$x_{l_d^e} - x_{l_d^s} \geq 1 \quad \text{for all dependencies } d \tag{4.2}$$

where $x_l \geq 0$.

# 5 Model 2 – Formulation, solution method and results

The variables are the same as in Model 1 except that we exclude the $x$-variables. The constraints are the same except that constraints (3.4) are replaced by

$$\sum_{d \in C} \delta_d \geq 1 \quad \text{where } C \text{ is a cycle of dependencies.} \tag{5.1}$$

Here (5.1) secures that we get no cycles in the virtual channel/dependency network.

We again use GLPK and use in principle the same search method that we used for Model 1. The number of constraints (5.1) is, however, vast. We therefore generate them on the fly as we need them during the solution process. Whenever we have selected a $\delta_d$ to be rounded to zero we check whether the dependence $d$ together with other other dependencies with $\delta$ rounded to zero forms cycles in the network. If it does, we augment the model with the corresponding cycle avoidance constraints and round $\delta_d$ to one and proceed. If not, we round $\delta_d$ to zero and proceed.

We have successfully found solutions for a 5x5 torus with one or two virtual channels per physical channel and where there are one CPU attached to each switch and a connection requirement for each ordered pair of CPU-s. It is worth noting that the number of (5.1) constraints generated is fairly low, namely 164 in the one-virtual-channel case and 1122 in the two-virtual-channel case. In the two-virtual-channel case it is known that there is a multi-shortest path solution which is deadlock-free. The deadlock-free solution that we found turned out to be a multi-shortest path solution.

We see from the model description that if we have $D$ dependencies involving switches only, $R$ requirements, and $L$ virtual channels per physical channel, the number of constraints is $(D + L + 2)R + D$ plus the number of generated cycle avoidance constraints which cannot be determined beforehand, and the number of variables is $DR + 2D$. It is the terms $DR$ and $LR$ which blow up the problem. This again motivated us to search for an alternative approach which scales better.

# 6 Model 3 – Notation and formulation

We introduce the following additional notation:

$M$ a large positive number

$p$ path of dependencies along which a requirement may be routed

Variables in the problem:

$v_{pr}$ flow along path $p$ covering requirement $r$, $\geq 0$
$\delta_d$ 0 if dependency $d$ is used, 1 otherwise
$s_r$ slack variables representing unsatisfied requirements, $\geq 0$

The mixed integer linear programming problem becomes:

$$\text{minimize} \sum_{p,r} l_p v_{pr} + M \sum_r s_r \tag{6.1}$$

subject to:

$$\sum_{p \in r} v_{pr} + s_r \geq 1 \quad \text{for all requirements } r \tag{6.2}$$

$$\sum_{r,p \in d} v_{pr} + t\delta_d \leq t \quad \text{for all not preset dependencies involving switches only} \tag{6.3}$$

$$\sum_{d \in C} \delta_d \geq 1 \quad \text{for all cycles } C \text{ of dependencies} \tag{6.4}$$

# 7 Model 3 – Solution method

The problem as formulated above has a huge number of $v_{pr}$-variables and cycle avoidance constraints (6.4). In order to obtain an approach which scales we generate on the fly during the solution process only $v_{pr}$-variables which are of current interest, and only cycle avoidance constraints which secure deadlock-free routings.

We use the same depth-first search as we used in Model 2. In every node in the search tree that we process we solve the LP with dynamic generation of $v_{pr}$-variables. We first solve the LP with the already generated $v_{pr}$-s and cycle avoidance constraints. As in the simplex algorithm we then try for each requirement to introduce into the problem a path variable $v_p$ which does not use dependencies $d$ with $\delta_d = 1$ and which has minimum relative cost. From linear programming basics we know that the relative cost $c_p$ of path variable $v_{pr}$ is

$$c_{pr} = (\sum_{d \in p(r)} (1 - \pi_d)) - \pi_r \tag{7.1}$$

where $\pi_r$ is the dual variable associated with constraint (6.2) and $\pi_d$ is the dual variable associated with constraint (6.3). We know further from linear programming basics that after the optimization $\pi_r \geq 0$ and that $\pi_d \leq 0$.

Finding a path variable covering requirement $r$ with minimum relative cost amounts to finding a shortest path from requirement $r$'s origin to its destination where the length of dependence arc $d$ is 1, and where we exclude using dependency arcs $d$ with $\delta_d$ fixed to one. If we find path variables $v_{pr}$ with

$c_{pr} < 0$ we introduce them into the problem, run the optimization again and recalculate the relative costs $\pi_r$. We continue in this way until we cannot any longer find path variables with negative relative costs.

The number of constraints is $D + R$ plus the number of generated cycle avoidance constraints. The number of $\delta_d$-variables is $D$. The number of path variables and cycle avoidance constraints is potentially enormous, but practice shows that only a tiny subset of path variables and cycle avoidance constraints will be generated during the solution process. So our experience is that Model 3 scales very well.

Using our algorithm we shall in principle either find a deadlock-free solution or conclude that no deadlock-free solution exists after a finite number of computations. Of course, if no solution exists, the running time will be unacceptable because of NP-completeness, so we shall always set an upper bound on the number of LP-s that will be run. In our experiments we have restricted ourselves to symmetric problems where a deadlock-free solution is known to exist. We have not been able to prove that in this case the problem of finding the shortest deadlock-free solution is NP-hard, but our guess is that it is.

# 8 Model 3 – Results

In experimenting with the models we have used the open source linear programming library GLPK installed on a Lenovo PC with an
Intel® Core ™ i7-3520M CPU @ 2.90GHz4 processor. We have limited ourselves to networks with one virtual channel per physical channel since these are the most difficult networks to find deadlock-free routings in. We have focused on two-dimensional and three-dimensional symmetric tori of switches with one CPU attached to each switch and where there is a connection requirement in both directions between all pairs of CPU-s.

**Two-dimensional tori**
To run the model with all the requirements included takes unacceptable long time. Therefore we proceeded as follows:

(i) We first ran the model by only including requirements between pairs of CPU-s attached to switches separated by two channels.

(ii) The dependencies used in (i) form an acyclic network. We assign numbers to the virtual channels as described in Section 4 and augment the subnetwork of dependencies used in (i) with all dependencies where the end virtual channel has a higher number than the start virtual channel. Then we ran the model with all requirements where we allowed dependencies in this extended subnetwork only.

(iii) If we experienced unsatisfied requirements in (ii), we ran the model with these unsatisfied requirements only where we fixed for use all the dependencies used in (ii) but allowed for additional dependencies to be used. If we still should get unsatisfied requirements (which we have not experienced in practice), we would unfix a fixed dependency with the smallest thruput and include all the requirements which use this dependency into the set of unsatisfied requirements and run the model again. We would continue in this way until we satisfy all the unsatisfied

| | **10x10 torus** | **11x11 torus** | **12x12 torus** |
|---|---|---|---|
| Run time (sec) | 144 | 257 | 225 |
| LP-s run | 478 | 396 | 306 |
| Cycle cuts | 1703 | 1596 | 826 |
| Sum of path lengths | 44274 | 73861 | 114346 |
| Relaxed solution | 40100 | 65340 | 103824 |
| Percent increase | 10 | 13 | 10 |
| | **13x13 torus** | **14x14 torus** | **15x15 torus** |
| Run time (sec) | 732 | 616 | 2045 |
| LP-s run | 558 | 412 | 752 |
| Cycle cuts | 2738 | 1148 | 4162 |
| Sum of path lengths | 176572 | 252456 | 370666 |
| Relaxed solution | 156156 | 230692 | 327713 |
| Percent increase | 13 | 9 | 13 |

Table 1: Results for two-dimensional tori

requirements.

(iv) Then we ran the model with all requirements where we allowed for use the depencencies used in (ii) and (iii).

(v) Finally we ran the model ignoring the cycle avoidance constraints. This is simply solving the multi-shortest-path problem. This was done in order to be able to calculate the percentage increase in the sum of path lengths for our solution compared to the sum of path lengths for the multi-shortest-path solution. This gives a conservative estimate of the difference between the minimal sum of path lengths with or without cycle avoidance since we know that in general the minimal path length sum will increase when we require deadlock avoidance.

The results are given in Table 1. Reported LP-s run and cycle cuts generated are reported for (i) and (iii). (ii) and (iv) are just shortest path calculations in acyclic networks.

| | **5x5x5 torus** | **6x6x6 torus** |
|---|---|---|
| Run time (sec) | 805 | 4653 |
| LP-s run | 826 | 1590 |
| Cycle cuts | 3440 | 10501 |
| Sum of path lengths | 44575 | 177594 |
| Relaxed solution | 40750 | 177594 |
| Percent increase | 9 | 9 |

Table 2: Results for three-dimensional tori

**Three-dimensional tori**
For an $n$x$n$x$n$ torus we proceeded as follows:

(i) We first ran the model by only including ($n$-1)x($n$-1)x($n$-1) requirements between pairs of CPU-s attached to switches separated by two channels.
(ii) Then we ran the model with all requirements between pairs of CPU-s attached to switches separated by two channels minus the requirements used in (i) where we fixed for use all dependencies used in (i) but allowed for additional dependencies to be used.
(iii) Then we ran the model with all requirements minus all requirements between pairs of CPU-s attached to switches separated by two channels where we allowed dependencies used in (i) and (ii) only.
(iv) If we experienced unsatisfied requirements in (iii), we ran the model with these unsatisfied requirements only where we fixed for use all the dependencies used in (i) and (ii) but allowed for additional dependencies to be used.
(v) Finally we ran the model ignoring the cycle avoidance constraints.

The results are given in Table 2.

# 9 Putting a common limit on the virtual channel thruputs

One problem with a deadlock-free routing is that some channels may be overused. Therefore we have augmented Model 3 with constraints which put a common upper bound on the traffic that can be channeled through a single virtual channel. The revised model consists of the objective function and all the constraints in Model 3 augmented with the following constraints:

$$\sum_{r,p \in l} v_{pr} \leq k \quad \text{for all virtual channels } l \tag{9.1}$$

As an example we have run this model for a 10x10 torus with one virtual channel per physical channel. The results are given in Table 3 where we use the procedure numbering used above for two-dimensional tori. Since we are are solving the problem in steps we have incorporated an upper bound in the relevant steps (i) and (ii). When we put an upper bound of 6 in step (i) the

| Thruput bound in (i) | Thruput bound in (ii) | Max thruput in (i) | Max thruput in final solution | Sum of path lengths |
|---|---|---|---|---|
| $\infty$ | $\infty$ | 8 | 405 | 44274 |
| 7 | 350 | 7 | 395 | 44232 |
| 7 | 300 | 7 | 373 | 44282 |
| 7 | 250 | 7 | 356 | 44112 |
| 7 | 200 | 7 | 366 | 44110 |
| 6 | - | - | - | - |

Table 3: Results with channel thruput bounds

model just runs and runs which indicates that the problem we try to solve in this step probably has no feasible solution.

# 10 Recovery after link fault

We shall here look into the problem where one physical channel fails in a network where we have a deadlock-free routing. We want to reroute the requirements such that we deviate as little as possible from the original routing. We propose the following algorithm:

(i) In the acylcic virtual channel/dependency network consisting of the dependencies used in the error-free routing we assign channel numbers as described in Section 4.
(ii) Remove from the topology and from the set of dependencies used in the error-free routing all dependencies which involve the failed channel.
(iii) Augment the remaining set of used dependencies with all dependencies which do not involve the failed channel and where the number of the end channel is larger than the number of the start channel. Run Model 3 where we allow the resulting set of dependencies only, and where we give a bonus to the dependencies used in the error-free routing.
(iv) If we experience uncovered requirements in (iii), we run the Model 3 for the set of uncovered requirements where we fix for use all requirements used in (iii), but allow for use all other error-free dependencies.
(v) If we still experience uncovered requirements, we unfix one of the dependencies fixed for use which has the lowest thruput, add those requirements which used it to the set of uncovered requirements, and repeat (iv). Continue in this way until all requirements are covered.

We have used this approach on the 10x10 torus as an example where we have removed as failed the most used channel in the existing deadlock-free routing which had pathlength sum 44274. The rerouting around the failed channel gave a solution with pathlength sum 45006.

# 11 Primal methods

What characterizes the heuristics we have described so far is that they are 'dual' in the sense that we iterate through a series of infeasible solutions until we hopefully arrive at a good feasible solution. The disadvantage is that available processing time may not be sufficient to reach a feasible solution and we are left with nothing. Therefore we should look for *primal* heuristics, i.e. heuristics where we from an initial (low quality) feasible solution iterate through a series of feasible solutions towards a 'local optimum' that the heuristic cannot improve upon.

One primal approach that comes to mind is to solve the problem for a subnetwork where it is easy to find a feasible solution and iteratively try to improve on this solution. For e.g. tori (complete or amputated) candidate subnetworks may be cylinders or grids. For general symmetric networks spanning trees may form initial subnetworks. We shall describe how we can iterate from any spanning tree solution. It is shown in [2] that that the problem of finding an optimal spanning tree solution is NP complete even if all the channels have equal length. Whether the problem is pseudopolynomial or not is not known to the author. We shall now describe the heuristic we propose. Since a spanning tree is in general a rather meager subtopology we cannot expect that we get very good solutions. Therefore we shall not treat this topic in detail. We shall focus on symmetric networks where each channel has one layer only and where the requirements are symmetric also. Here is the heuristic we have implemented.

1 Fix an initial spanning tree in the undirected switch/channel network, for example an Up*/Down* spanning tree and calculate the objective function for the tree.

2 Select an out-of-tree channel which has not been selected earlier and include it tentatively in the spanning tree. This channel will form a circle together with in-tree channels.

3 Remove tentatively one by one of the in-tree channels in the circle so that we find a series of revised spanning trees. For each revised spanning tree calculate the objective function. Save the best revised spanning tree.

4 Go to 2 and continue until we have examined all out-of-tree channels and save the revised spanning tree with the best value of the objective function.

5 When we have exhausted all out-of-tree channels we go to 1 where we have replaced the initial spanning tree with the best spanning tree we have found so far. Continue in this way until we do not experience any improvement.

6 Replace the undirected channels in the final tree by two directed channels, establish the corresponding acyclic channel/dependency network and assign channel numbers as described earlier.

7 Finally augment the network found in 6 with all dependencies where the end channel has a higher number than the start channel and route all requirements along shortest paths in this acyclic network.

This heuristic is not guaranteed to give an optimal tree when we reach step 5. A simple example that shows that the tree can be suboptimal is the network with requirement 1 between all pairs of switches and with channel lengths equal to 1 and where the switched are numbered 1, 2, 3, 4, 5, 6, and the channels are (1,2), (2,3), (3,4), (4,5), (4,6), (2,5), (2,6). The initial tree is (1,2), (2,3), (3,4), (4,5), (4,6) with value 29. Symmetry implies that we can limit ourselves to consider (2,5) as the out-of-tree channel to be introduced to create the next tree. The alternative in-tree channels to be removed in order to create the next tree are (4,5), (2,3) and (3,4). This gives three new trees with values 32, 32, and 29 which means that the heuristic terminates with the initial tree. The optimal tree, however is (1,2), (2,3), (3,4), (2,5), (2,6) with value 28.

# 12 Coupling together two or more symmetric networks with known deadlock-free routings

We propose the following algorithm for the coupling together of n networks with known deadlock-free routings:

(i) For the combined virtual swith/channel network make an undirected network by identifying the channels $(s, s')$ and $(s', s)$.
(ii) Find the longest spanning tree in this undirected network where the length of each symmetrized dependency is set equal to the sum of the utilizations of the two components of the symmetrized channel.
(iii) Expand the undirected network back to the corresponding directed network where the spanning tree will expand to a directed acyclic subnetwork, create the corresponding acyclic channel/dependecy network, assign channel numbers as described earlier, and augment the acyclic subnetwork with all dependencies where the end channel has a higher number than the start channel.
(iv) Route all requirements along shortest paths in this extended acyclic network.

In (iv) it is of course possible to give a bonus to those dependencies used in the deadlock-free routings in the component networks dependent on how much they were used.

As a small experiment we coupled together two 5x5 two-dimensional tori with one CPU associated with each switch and with one two-way bridge channel. We had a requirement of 1 between every ordered pair of CPU-s. If we did not use the expansion described in (iv), we obtained a total usage of dependencies equal to 12904 and used 140 dependencies. When we used the expansion described in (iv), we obtained a total usage of dependencies equal to 10512 and used 329 dependencies. Of course, this approach is polynomial, and can be run for a high number of large component networks with any set of bridging channels in reasonable time.

In lieu of going via the undirected network in (i), one could try to find a maximum acyclic network directly. This problem is NP complete [3] and good approximate algorithms are hard to find [4].

# 13 Final remarks

Our input format also allows for the registration of (geographical) areas together with area restrictions on the paths for each individual requirement, a service level for each individual requirement together with a specification of layers allowed for each service level, bandwidth capacities for the individual ports, an upper bound on the number of layers each individual port can accommodate, and bandwidth specification for each individual requirement.

# 14 Acknowledgements

The author wants to thank O.Lysne, S.A.Reinemo, and T.Skeie for helpful discussions and suggestions. The method used in Model 3 is well-known and is called 'branch-and-cut-and-generate'. The challenge has been to write the software needed for our problem. The author wants to thank Y.Halbwachs, T.V.Stensby, Ø.Hjelle, and T.Dreibholz for all the time and energy they have spent helping me master the intricacies of advanced C++ programming and incorporating the use of the Boost library.

# 15 References

[1] S.Toueg and K.Steiglitz, *Some complexity results in the design of deadlock-free packet switching networks*, SIAM Journal of Computing, Vol.10, No.4, November 1981.

[2] D.S.Johnson, J.K.Lenstra and A.H.G. Rinnoy Kan, "The complexity of the network design problem", Networks, Vol.8, 1981, pp 279-285.

[3] R.M.Karp, "Reducibility among combinatorial problems", pp. 85-103 in *Complexity of Computer Computations* R.E.Miller and J.W.Thatcher, Eds.Plenum, New York, 1972.

[4] V.Guruswami, R.Manokaran, P.Raghavendra, "Beating the Random Ordering is Hard: Inapproximability of Maximum Acyclic Subgraph", https://www.cs.cmu.edu/ venkatg/pubs/papers/mas.pdf