Graph optimization approaches for minimal rerouting in symmetric three stage Clos networks

Kaj Holmberg

Kaj Holmberg, Graph optimization approaches for minimal rerouting in symmetric three stage Clos networks, 2009, Journal of Mathematical Modelling and Algorithms, (8), 1, 81-100.
http://dx.doi.org/10.1007/s10852-008-9090-0

Journal of Mathematical Modelling and Algorithms manuscript No.
(will be inserted by the editor)

Graph Optimization Approaches for Minimal Rerouting in Symmetric Three Stage Clos Networks

Kaj Holmberg

September 12, 2008

Abstract We consider routing in symmetrical three stage Clos networks. Especially we search for the routing of an additional connection that requires the least rearrangements, i.e. the minimal number of changes of already routed connections. We describe polynomial methods, based on matchings and edge colorings. The basic idea is to swap colors along alternating paths. The paths need to be maximal, and the shortest of these maximal paths is chosen, since it minimizes the rerouting that needs to be done. Computational tests confirm the efficiency of the approach.

Keywords Clos networks, switches, edge coloring, matching, optimization

1 Introduction

In this paper we study how to route connections through symmetrical three stage switching networks of Clos type [4]. The case of simultaneous routing of a number of connections is well treated in the literature, but we focus on the following case. Assume that a number of connections already are routed, and that a new connection shall be routed. If possible, we route the new connection without changing any previously routed connection. If this is not possible, we wish to change as few as possible of the previously routed connections in order to be able to route the new one.

There are several reasons why minimal rerouting may be important. In practice, connection requests usually do not appear all at the same time. If the rerouting involves some manual or relatively time consuming activity, one would like to minimize the work/time needed. Furthermore, if rerouting gives a short break in the rerouted connection, as few connections as possible should be disturbed.

We use graph optimization approaches to construct a polynomial method that finds the optimal solution, i.e. a routing with the minimal number of rearrangements.

In section 2, we describe the problem and give a general mathematical model. In section 3, we view the problem in terms of matchings and edge coloring, and show how routing from scratch can be done in polynomial time. In section 4, we discuss the problem of routing to achieve minimal rerouting in heuristic terms. In section 5, we

Department of Mathematics, Linköping Institute of Technology, SE-581 83 Linköping, Sweden
E-mail: kahol@mai.liu.se

use color swapping along a maximal alternating path, in order to obtain an optimal and polynomial method for finding solutions with minimal rerouting. In section 6, we describe a somewhat different strategy for finding the shortest maximal path. In section 7, we show that a swap between three colors can not produce a shorter maximal path than a swap between two colors. In section 8, we present the computational tests, and in section 9 draw some conclusions.

2 A mathematical model

We consider square crossbar switches, each with n inputs and n outputs, where any input can be connected to any output. These switches are organized in stages. Each stage is a column of switches, and each output of a switch in a stage is connected to an input of a switch in the next stage.

A connection is a request to connect a certain (left-most) input to a certain (right-most) output, and needs to be assigned a path through the network, using one switch in each stage.

Assuming that there are already a number of connections assigned through the network, a question is if a new request between an unused left-most input and an unused right-most output can be satisfied. If no path for a requested connection can be found, the network is called "blocking". If this never happens, the network is called "nonblocking". In a "strict-sense nonblocking" network, no existing connections need to be rerouted, but in a "rearrangeably nonblocking" network, some existing connections may need to be rerouted in order to allow the new one to be routed.

In general Clos networks [4], there are three stages, the first one with r $n \times q$ switches, the second one with q $r \times r$ switches and the third one with r $q \times n$ switches. Thus there are nr different inputs and the same number of outputs. If $q \geq 2n-1$, one can show that a Clos network is strict-sense nonblocking [4]. According to the Slepian-Duguid theorem [1], a Clos network is rearrangeably nonblocking if $q \geq n$. In this paper we focus on Clos networks based on symmetrical and identical $n \times n$ switches. We thus have $r = q = n$, so such a network is rearrangeably nonblocking, but not strict-sense nonblocking.

There are n switches in each of the three stages, yielding n^2 inputs and n^2 outputs. The interconnections between the stages are fixed as pictured for $n = 3$ in figure 1. There are never two connections between a pair of switches.

We assume that m specific connections are requested. For connection l there is a given input, the origin, o_l, and a given output, the destination, d_l.

In [7], we present a new model for this situation, based on the following notation. Connection l will in stage t use switch k_l^t, input i_l^t, and output j_l^t. Letting $N = \{0, \ldots, n-1\}$, we have $k_l^t \in N$, $i_l^t \in N$, and $j_l^t \in N$.

Knowing o_l and d_l, we can calculate what switches in the first and third stage that will be used, $k_l^1 = \lfloor o_l/n \rfloor$ and $k_l^3 = \lfloor d_l/n \rfloor$. Furthermore we have $i_l^1 = o_l - k_l^1 n$, and $j_l^3 = d_l - k_l^3 n$. The fixed couplings between the first and the second stage yields $k_l^2 = j_l^1$ and $i_l^2 = k_l^1$. The couplings between the second and third stage yields $k_l^2 = i_l^3$ and and $j_l^2 = k_l^3$. The choice to be made for connection l is thus only k_l^2.

In the mathematical model, we use the variables $x_{il} = 1$ if connection l uses switch i in the second stage. (This means that $x_{il} = 1$ for $i = k_l^2$, while $x_{il} = 0$ for all $i \neq k_l^2$.) Let $l \in C_O$ denote the "old" connections that were already routed, and $l \in C_N$ the new connections, to be routed. We assume that $|C_O| + |C_N| \leq n^2$, so all connections can

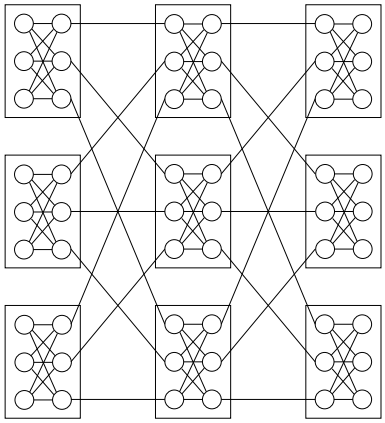

Fig. 1 3-stage switch.

be routed. Given a binary solution, $\bar{x}$, for $l \in C_O$, we let $A_1 = \{(i,l), l \in C_O : \bar{x}_{il} = 1\}$ and $A_0 = \{(i,l), l \in C_O : \bar{x}_{il} = 0\}$. A measure of the reroutings is given by

$$\sum_{(i,l)\in A_1} (1 - x_{il}) + \sum_{(i,l)\in A_0} x_{il} = \sum_{(i,l)} c_{il}x_{il} + |C_o|$$

where $c_{il} = -1$ for $(i,l) \in A_1$, $c_{il} = 1$ for $(i,l) \in A_0$ and $c_{il} = 0$ for all i and $l \in C_N$. A mathematical model for finding a feasible routing that minimizes the number of reroutings is given below.

$$\begin{aligned}
v^* = \max & \sum_{(i,l)} c_{il}x_{il} \\
\text{s.t.} & \sum_{i\in N} x_{il} = 1 \qquad l = 1,\dots m \qquad (1.1) \\
& \sum_{l\in L_k^1} x_{il} \le 1 \qquad i \in N, k \in N \qquad (1.2) \qquad \text{[P1]} \\
& \sum_{l\in L_k^3} x_{il} \le 1 \qquad i \in N, k \in N \qquad (1.3) \\
& x_{il} \in \{0,1\}\ \forall i,l
\end{aligned}$$

where $L_k^1 = \{l : k_l^1 = k\}$ (the set of connections that use switch k in the first stage) and $L_k^3 = \{l : k_l^3 = k\}$ (the set of connections that use switch k in the third stage).

P1 has mn variables and $2n^2 + m$ constraints (not counting the binary requirements). Constraints 1.1 ensure that each connection is routed (once), i.e. that each connection uses exactly one switch in the second stage. Constraints 1.2 ensure that there is at most one connection using switch k in the first stage and switch i in the second stage. Constraints 1.3 ensure that there is at most one connection using switch k in the third stage and switch i in the second stage.

We know that $-|C_O| \le v^* \le |C_O|$, and the number of reroutings will be equal to $n_r = (v^* + |C_O|)/2$. Even though P1 allows for several new connection requests, we belive that in practice the new requests will usually appear one at a time. Therefore we will in the rest of this paper focus on the case where $|C_N| = 1$.

The simultaneous routing problem is to find any feasible solution to P1, i.e. ignoring the objective function.

l	o_l	d_l	k_l^1	k_l^3	k_l^2	Path
1	0	1	0	0	0	0-0, 0-0, 0-1
2	1	4	0	1	1	1-1, 3-4, 4-4
3	2	2	0	0	2	2-2, 6-6, 2-2
4	3	7	1	2	2	3-5, 7-8, 8-7
5	6	0	2	0	1	6-7, 5-3, 1-0
6	7	5	2	1	0	7-6, 2-1, 3-5

Table 1 A small example.

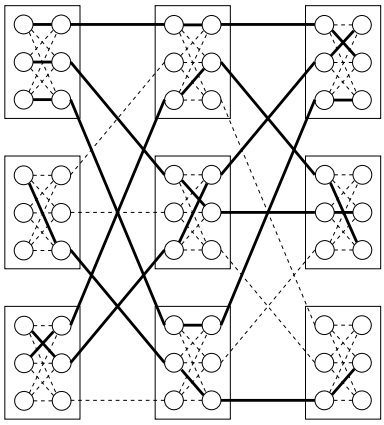

Fig. 2 Connections in 3-stage switch.

Let us give a small example for the case where $n = 3$. Assume that $m = 6$, and the origins and destinations are given in table 1. The table also gives a solution obtained by solving P1, in the form of k_l^2. The solution can be expanded into a full path for each connection. The paths are illustrated in figure 2. An example of adding a new request will be given later.

3 Matchings and edge colorings

Now we consider the bipartite multigraph of the switches in the first stage and third stage, where connection l corresponds to the edge from node k_l^1 in the first level to node k_l^3 in the second. Further consider all connections that use a specific switch, i, in the second stage, denoted by L_i^2. At most one connection is allowed from each switch in stage 1, see constraints 1.2, and at most one connection is allowed to each switch in stage 3, see constraints 1.3. This can be seen as a *matching* in the bipartite multigraph, see [8] and [3].

Consider again the example in the previous section. In table 1 the data, including k_l^1 and k_l^3, are given. In figure 3, the connections between k_l^1 and k_l^3 are given. Note that this information is indata. Here we have $L_0^1 = \{1, 2, 3\}$, $L_1^1 = \{4\}$, $L_2^1 = \{5, 6\}$, $L_0^3 = \{1, 3, 5\}$, $L_1^3 = \{2, 6\}$, and $L_2^3 = \{4\}$.

The solution given in table 1 and figure 2 gives information of which switch in the second stage each connection uses. Here we have $L_0^2 = \{1, 6\}$, $L_1^2 = \{2, 5\}$, and $L_2^2 = \{3, 4\}$. This tells us how to split up the total solution in figure 3 into three

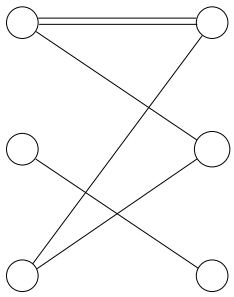

Fig. 3 Connections pictured in a bipartite graph.

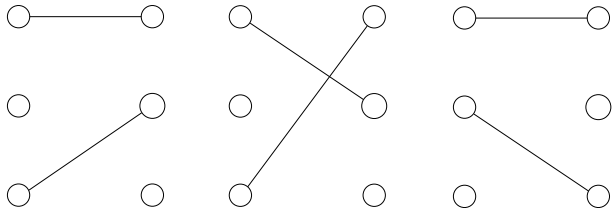

Fig. 4 Connections split up into three matchings.

matchings, one for each switch in the second stage. These three matchings are shown in figure 4. Note that the value of k_l^2 tells us which matching connection l belongs to.

Let us relate this to P1. We can rephrase the variable definition somewhat. Since each switch in the center stage is associated with a certain matching, we have $x_{il} = 1$ if connection l, i.e. edge (k_l^1, k_l^3), is included in matching i. Constraints (1.2) and (1.3) then ensure that we get a matching for each i. Constraints (1.1) ensure that each connection is included in one matching. We can here note that $|L_k^1|$ is the number of edges that are adjacent to switch k in stage 1, and $|L_k^3|$ is the number of edges that are adjacent to switch k in stage 3. It is not difficult to prove the following.

Lemma 1 *All connections in a matching can be done via the same switch in the second stage. Any feasible solution to P1 corresponds to the union of a number of matchings, one for each switch in the second stage. Any union of at most n matchings corresponds to a feasible solution to P1.*

Thus a collection of matchings constitutes a solution. Deciding how to split the multigraph up into matchings yields a solution to P1, and can be done as follows.

Algorithm 1:

1. Construct the bipartite multigraph, $G(N, N, E)$ where $E = \{(k_l^1, k_l^3), l = 1, \ldots, m\}$. Let $S = N$ be the set of available switches (in stage 2).
2. Find a maximal cardinality matching, M, in G.
3. Route the connections corresponding to each edge in M through a switch $i \in S$.
4. Let $S = S \setminus \{i\}$. (Remove the used switch.)
5. Let $E = E \setminus M$. (Remove the routed connections.)
6. If $E = \emptyset$, stop. Otherwise go to 2.

Note that $|E| = m$ in step 1. It is clear from the lemma above that this algorithm produces a feasible solution to P1. The procedure only stops when all edges are treated, i.e. all connections are routed.

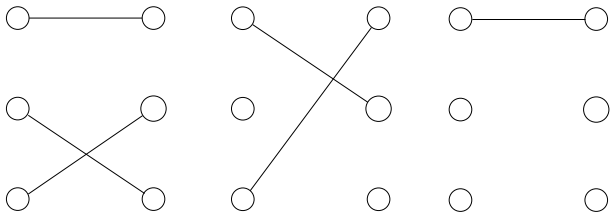

Fig. 5 Maximal matchings.

It is observed in [5] that it is not necessary to use a maximal matching. It is sufficient to use matchings that cover all nodes of maximal degree. The maximal degree in the graph will be

$$d = \max(\max_k |L_k^1|, \max_k |L_k^3|).$$

If a matching including all nodes with this degree is found and removed, the maximal degree in the remaining graph will be $d-1$. Thus this only need to be repeated d times. Clearly $d \leq n$, so there will be at most n matchings. Since there are n switches in the center stage, each matching can be associated with one switch. Thus a feasible solution is found in at most n iterations.

This leads to the method of *edge coloring*, where each matching is colored with a specific color. The problem is to color all the edges so that no two edges adjacent to the same node has the same color. Obviously d colors will be sufficient [2], so we need only d switches in the center stage.

The variable definition can now be formulated as $x_{il} = 1$ if edge (k_l^1, k_l^3) is colored by color i. In other words edge (k_l^1, k_l^3) is colored by color k_l^2.

In [5] it is shown how to find a minimal edge coloring in a bipartite graph in $O(|E| \log |V|)$. (Note that "minimal" here refers to the number of colors/switches, which is different from the objective function of P1.) More recent methods yield $O(|E|d)$ in [11] and $O(|E| \log d)$ in [6]. In our graph $|E| = m$ and $|V| = 2n$, which yields $O(m \log n)$, $O(md)$ and $O(m \log d)$. Furthermore $d \leq n$, so $O(md) = O(mn)$ and $O(m \log d) = O(m \log n)$. Assuming that $m = O(n^2)$, the complexity of the edge coloring method is $O(n^2 \log n)$. This proves that the problem of finding a simultaneous routing of at most $m \leq n^2$ connections is polynomially solvable. On the other hand, if $m > n^2$ there is no feasible routing.

Lemma 2 *Simultaneous routing through a three stage Clos network can be done in polynomial time.*

Now consider the previous example. In step 1 of the algorithm, the graph in figure 3 is created. A maximal matching is for example $M = \{(0,0),(1,2),(2,1)\}$. These edges correspond to connections 1, 4 and 6, so these connections are routed through switch 0 in stage 2. Removing these edges, we get a maximal matching $M = \{(0,1),(2,0)\}$, which corresponds to connections 2 and 5. Thus these connections are routed through switch 1 in stage 2. Removing these edges leaves only $M = \{(0,0)\}$, which corresponds to connection 3, which thus is routed through switch 2 in stage 2, and we are finished. The matchings can be seen in figure 5.

Considering edge coloring, we find that the maximal degree in G is 3, and that node 0 in stage 1 and node 0 in stage 3 have degree 3. Therefore a matching covering these nodes is assigned the first color. One such a matching is $M = \{(0,0)\}$. Removing this

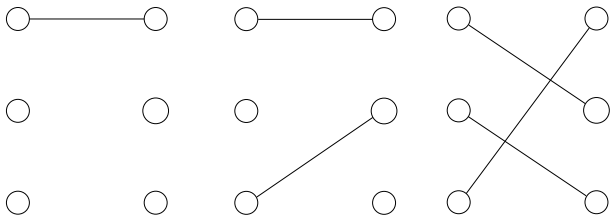

Fig. 6 Edge coloring.

matching yields a graph where the maximal degree is 2. Nodes 0 and 2 in stage 1 and nodes 0 and 1 in stage 3 have degree 2, so a matching covering these nodes is assigned the second color. One example of such a matching is $M = \{(0,0),(2,1)\}$. Removing this matching yields a graph where all nodes have degree one. We get the matching $M = \{(0,1),(1,2),(2,0)\}$, see figure 6.

4 Minimal rearrangements

We now assume that there are already $m-1$ connections set up, and a new one is requested. We assume that $m \leq n^2$, since otherwise it will be impossible to route another connection. We also assume that the input o_m and the output d_m both are free. From o_m and d_m we calculate k_m^1 and k_m^3, and look for a path between these switches.

Trying to route a connection through switch k in the first stage and switch i in the second stage, we find that it can be done if $L_i^2 \cap L_k^1 = \emptyset$. Otherwise $L_i^2 \cap L_k^1$ is the connection that use/block this route. Similarly, $L_i^2 \cap L_k^3$ is the connection that use/block the route through switch i in the second stage and switch k in the third stage. If this set is empty, the route is free and can be used.

If there exists some i such that $L_i^2 \cap L_{k_m^1}^1 = \emptyset$ and $L_i^2 \cap L_{k_m^3}^3 = \emptyset$, then switch i in the second stage can be used for routing connection m, without rearranging any existing connections. If there is no such i, then one or more existing connections must be rerouted, in order to accommodate the new connection requirement.

There are direct heuristic ways of doing this, as discussed in [7]. There is however a risk of cycling, so care must be taken when choosing connections to reroute. There are several heuristics proposed in the literature, see for example [9]. There one works with partly infeasible solutions, where paths may end up at the wrong switch in stage 3. One then applies a swap heuristic that iteratively corrects this. A fairly elaborate scheme seems necessary in order to avoid cycling. A number of proposed methods of this type have subsequently been shown to fail in certain circumstances.

Let us now use the matching approach, described above. Consider the previous example, where we add a request for a new connection, with $o_7 = 4$ and $d_7 = 3$. The solution is pictured in figure 3, and can be partitioned into three matchings, see figures 4, 5 and 6 for three different solutions.

The new requested connection has $k_7^1 = 1$ and $k_7^3 = 1$. In figure 5, these nodes are free in matching 2, so the new edge can be added to it. In figure 6, the new edge can be added to matching 0. However, in figure 4 we find that in none of the matchings both these nodes are free, so the edge (1,1) can not be inserted in any of the matchings. It is possible to move the edge (1,2) in matching 2 to matching 0, since these two nodes

are free. We then get the matchings in figure 5, and can add the new edge to the last matching.

Moving an edge from one matching to another corresponds to moving a connection from one second stage switch to another. We can device more complicated algorithms for heuristically swapping edges until the new one can be inserted. Again there is a danger of cycling, and furthermore we cannot be sure that a solution with minimal rearrangements is found. However, in the following section we describe a method which is guaranteed to produce a minimal rearrangement solution.

5 Color swapping

Rearrangements using edge coloring can be done systematically as follows. Assume that we have an edge coloring, and consider only two of the colors. Temporarily we may draw parallels to ordinary matching by considering the edges colored by the first color as matched, the edges colored by the second color as unmatched, and ignoring all the edges colored with other colors.

We recall that an alternating path is a path along which every second edge is matched (and the others are not), and an augmenting path is an alternating path which starts and ends with unmatched edges. It is well-known that a matching is maximal if and only if there does not exist an augmenting path. If there exists an augmenting path, then the cardinality of the matching can be increased by swapping matched and unmatched edges (i.e. letting all previously unmatched edges be matched and all previously matched edges be unmatched).

Here we let an "alternating path" be a path along which the edges are of two alternating colors. Furthermore, let a "maximal alternating path" be one that cannot be extended, i.e. starts and stops at nodes with only one of the two colors adjacent. A "swap" here means replacing each color with the other one. We recall that the edges of a certain color in an edge coloring forms a matching.

Lemma 3 *Swapping the two colors along a maximal alternating path in an edge coloring produces another feasible edge coloring.*

Proof The first color forms a matching, and after a swap, it is clearly still a matching. The same is true for the matching corresponding to the second color. The other colors are not affected by the swap. Thus we still have an edge coloring after the swap.

Clearly this does not hold if the path is not maximal. By finding a maximal alternating path and swapping the colors along it, we have a way of changing which colors are adjacent to certain nodes. This can be used in order to "free" a node of one color, and thereby enable inserting a new edge.

This idea is described in [3], and it can be used to prove Konig's theorem, with the so called Kempe-chain argument. In that context it is used to construct a routing from scratch, and not to minimize the rearrangements. [3] contains a general description and comparison of different techniques for constructing routings from scratch. Here we will use this idea in order to find the routing with minimal rearrangements.

Let us label the nodes with the colors of all adjacent edges as follows. Let C_i^{NL} be the set of colors adjacent to node i in the left-most level and C_j^{NR} be the set of colors adjacent to node j in the right-most level. Furthermore, let $M^R(c, i)$ be the node in the right-most level connected to node i (in the left-most level) via an edge

of color c. If there is no edge of color c adjacent to node i, we let $M^R(c,i) = -1$. Similarly, let $M^L(c,j)$ be the node in the left-most level connected to node j (in the right-most level) via an edge of color c, or -1 if there is none. This means that $M^R(k_l^2, k_l^1) = k_l^3$ and $M^L(k_l^2, k_l^3) = k_l^1$. We note that $C_j^{NL} = \{c : M^R(c,j) \geq 0\}$ and $C_i^{NR} = \{c : M^L(c,i) \geq 0\}$, so C^{NL} and C^{NR} are strictly speaking not necessary to use, but it is convenient to keep them in the discussion.

Now assume that we wish to insert an edge between node i and node j. If there is any color (used for other edges) absent from the labels of both node i and j, the new edge can be colored with this color. Let C be the set of colors that may be used. If there exists some $k \in C \setminus (C_i^{NL} \cup C_j^{NR})$, then we color the new edge with k. This means that no other connection needs to be rerouted in order for the new connection to be routed.

Let us now assume that this is not the case, i.e. that all existing colors are used for edges adjacent to either i or j or both (i.e. that $C_i^{NL} \cup C_j^{NR}$ is the set of all used colors).

If each existing color is included in C_i^{NL} or in C_j^{NR} (i.e. all colors are adjacent to one of the nodes), then it is not possible to use any existing color for the new edge, so a new color must be used. If the maximal number of colors is already used (i.e. the maximal number of switches), then it is not possible to add the new edge (i.e. it is not possible to route the new connection).

The only remaining possibility is that there exists one color, c_A, that is not in C_j^{NR}, and one color, c_B, that is not in C_i^{NL}. Then we have $c_A \in C_i^{NL}$ and $c_B \in C_j^{NR}$, so $c_A \in C_i^{NL} \setminus C_j^{NR}$ and $c_B \in C_j^{NR} \setminus C_i^{NL}$. In words, there is one edge of color c_A adjacent to node i but none adjacent to node j, and one edge of color c_B adjacent to node j but none adjacent to node i.

Lemma 4 *Exactly one of the three cases below occurs for each pair* (i,j).

I. $C \setminus (C_i^{NL} \cup C_j^{NR}) \neq \emptyset$.

II. $C = C_i^{NL}$ *or* $C = C_j^{NR}$.

III. $C_i^{NL} \setminus C_j^{NR} \neq \emptyset$ *and* $C_j^{NR} \setminus C_i^{NL} \neq \emptyset$.

In case I, we can use a free color for the edge (i,j). In case II, we need to use a new color. In case III, we can enable coloring the edge (i,j) with an existing color as follows.

We find a simple path alternating between colors c_A and c_B, starting at node i with an edge of color c_A. The path should be maximal, i.e. if it ends with an edge of color c_A (c_B), there should be no edge of color c_B (c_A) adjacent to the ending node.

The ending node will be different from j (as is shown below). If we swap the two colors along the path, there will be no edge of color c_A adjacent to node i or node j, so the new edge can be colored with c_A.

A single edge of color c_A is sufficient for doing a swap. The number of edges in the path is equal to the number of swaps, which is equal to the number of rerouted connections, so the *shortest* maximal path will yield the minimal number of rearrangements.

Assuming that node i is in the first (leftmost) stage, node j will be in the rightmost. The alternating path will start with an edge with color c_A, and go from left to right. The second edge will be of color c_B and go from right to left. This pattern will be repeated, so clearly all edges of color c_A included in the alternating path will go from left to right, and all edges of color c_B included in the path will go from right to left.

If the path was to end at node j, it would have to end with an edge of color c_B used from left to right, but this cannot happen. The path will consequently never end at j.

From this reasoning we can also conclude that the path can never return to node i, since there is no edge of color c_B adjacent to node i. The path can actually never return to an already visited node, since there is at most one edge of each color adjacent to a node. Thus the path will never contain a cycle. (Doing a swap along a cycle would not help, since no node would be freed of any color.)

Since there is at most one edge of a certain color adjacent to any node, the choice of next edge will be uniquely determined at each node. There is exactly one edge of color c_A adjacent to node i, and it leads to node $k = M^R(c_A, i)$. Node k will have at most one adjacent edge of color c_B, and so on.

Thus the whole path is uniquely determined, and is certain not to end up at node j or yield a cycle. The path will contain at least one edge. This tells us that the path can *always* be used for swapping colors and thereby enable the coloring of the new edge with c_A.

Lemma 5 *For each pair of colors $c_A \in C_i^{NL} \setminus C_j^{NR}$ and $c_B \in C_j^{NR} \setminus C_i^{NL}$, there exists one unique maximal alternating path, starting at node i, containing at least one edge, not including node j, and not containing a cycle.*

The only question that now remains is what two colors should be chosen. In order to find the minimal number of rearrangements, we must find the shortest maximal path. In order to achieve this, we can try all pairs of colors. We noted above that it is sufficient with a path consisting of only one arc, and obviously this is the shortest path that may exist. Therefore, if we find a path of length one, we can stop, and use that path. This corresponds to rerouting only one connection.

Let us describe the method for finding one alternating path in an algorithmic form. We start with a feasible solution (i.e. a feasible edge coloring) for connections $1, \ldots m-1$, and wish to add connection m with the least number of reroutings. (Assume that $c_A \in C_{k_m^1}^{NL} \setminus C_{k_m^3}^{NR}$ and $c_B \in C_{k_m^3}^{NR} \setminus C_{k_m^1}^{NL}$.)

Algorithm Swap2(c_A,c_B,m)

1. Let $K = 0$.
2. Let $i_1 = k_m^1$.
3. Set $j_1 = M^R(c_A, i_1)$. If $j_1 < 0$, go to 7.
4. Let $K = K + 1$.
5. Set $i_2 = M^L(c_B, j_1)$. If $i_2 < 0$, go to 7.
6. Let $K = K + 1$. Set $i_1 = i_2$ and go to step 3.
7. An alternating path is found. We may swap colors c_A and c_B along the path and set $k_m^2 = c_A$. Terminate.

At termination, K will be the length of the alternating path, i.e. the number of connections that need to be rerouted. In step 3, there will always be a $j_1 \geq 0$ in the first iteration. It can alternatively be formulated as finding l such that $k_l^1 = i_1$, $k_l^3 = j_1$ and $k_l^2 = c_B$. (We have $\{l\} = L_{i_1}^1 \cap L_{c_B}^2 \cap L_{j_1}^3$.) In step 5, the choice of i_2 can alternatively be formulated as finding l such that $k_l^1 = i_2$, $k_l^3 = j_1$ and $k_l^2 = c_A$. (We have $\{l\} = L_{i_2}^1 \cap L_{c_A}^2 \cap L_{j_1}^3$.) Note that Swap2 cannot fail (if $c_A \in C_{k_m^1}^{NL} \setminus C_{k_m^3}^{NR}$ and $c_B \in C_{k_m^3}^{NR} \setminus C_{k_m^1}^{NL}$). It always finishes with a usable path. If $K = 1$, i.e. we have a path with one edge, there is no need to continue searching for a shorter path.

Considering the complexity of this algorithm, the length of the path is $O(n)$. Each iteration has complexity $O(1)$, so the complexity of algorithm Swap will be $O(n)$.

In order to find the shortest alternating path, we have to run Swap for all different pairs of colors c_A and c_B. We start by calculating C_i^{NL} and C_j^{NR} for all i and j. Then we need the sets $C_{k_m^1}^{NL} \setminus C_{k_m^3}^{NR}$ and $C_{k_m^3}^{NR} \setminus C_{k_m^1}^{NL}$, since we will pick the two colors from these sets. In our implementation we will use the sets M^R and M^L rather than C^{NL} and C^{NR}, so we note the following. We have $C_i^{NL} = \{c : M^R(c,i) \geq 0\}$ and $C_j^{NR} = \{c : M^L(c,j) \geq 0\}$, so $C_i^{NL} \setminus C_j^{NR} = \{c : M^R(c,i) \geq 0, M^L(c,j) < 0\}$, and $C_j^{NR} \setminus C_i^{NL} = \{c : M^R(c,i) < 0, M^L(c,j) \geq 0\}$. Let us introduce the notation $C_{ij}^A = C_i^{NL} \setminus C_j^{NR}$ and $C_{ji}^B = C_j^{NR} \setminus C_i^{NL}$, and give the following short algorithm for creating these sets.

Algorithm CreateSets(i, j)

1. Set $C_{ij}^A = \emptyset$ and $C_{ji}^B = \emptyset$.
2. For each color $c \in C$
   (a) If $M^R(c,i) \geq 0$ and $M^L(c,j) < 0$, add c to C_{ij}^A.
   (b) If $M^R(c,i) < 0$ and $M^L(c,j) \geq 0$, add c to C_{ji}^B.

Clearly $|C_{ij}^A| \leq d$ and $|C_{ji}^B| \leq d$. If any of these sets is empty, we have $C_{k_m^1}^{NL} \subseteq C_{k_m^3}^{NR}$ or $C_{k_m^3}^{NR} \subseteq C_{k_m^1}^{NL}$, and can not do a swap. Before searching for an alternating path, however, we should check if there is a free color available that may be used directly, without swapping. (In this case, no rearrangement needs to be done in order to route the new connection.) If there exists a color $c_A \in C \setminus (C_{k_m^1}^{NL} \cup C_{k_m^3}^{NR})$, we can immediately set $k_m^2 = c_A$. Such a color has the property that $M^R(c_A, k_m^1) < 0$ and $M^L(c_A, k_m^3) < 0$. We use the following main algorithm.

Algorithm Swap(m)

1. If there exists a color such that $M^R(c_A, k_m^1) < 0$ and $M^L(c_A, k_m^3) < 0$, set $k_m^2 = c_A$. Terminate.
2. Call CreateSets(k_m^1, k_m^3) in order to get $C_{k_m^1,k_m^3}^A$ and $C_{k_m^3,k_m^1}^B$. If any of these sets is empty, add a new color, c_C, and set $k_m^2 = c_C$. Terminate.
3. For all pairs of colors $c_A \in C_{k_m^1,k_m^3}^A$ and $c_B \in C_{k_m^3,k_m^1}^B$, call Swap2($c_A$,$c_B$,$m$). If $K = 1$, terminate the loop.

If, in step 2, there is no free color, we cannot route the new connection. However, this only happens when $m > n^2$, and this is checked in advance.

We see that this algorithm calls Swap2 at most $|C_{k_m^1,k_m^3}^A||C_{k_m^3,k_m^1}^B|$ times, i.e. at most $O(d^2)$ times. (In practice it will usually be much less than d^2 times.) Thus we can find the shortest alternating path in not more than $O(nd^2)$ (i.e. $O(n^3)$). In other words, we can find a solution with minimal number of rearrangements in polynomial time.

Theorem 1 *Given a feasible routing, we can in polynomial time route a new connection with minimal number of reroutings of other connections (or decide that it cannot be routed).*

Let us consider the small example with the indata given in table 1. We start with the following solution, using colors A, B and C. $L_A^2 = \{1, 6\}$, $L_B^2 = \{2, 5\}$, $L_C^2 = \{3, 4\}$.

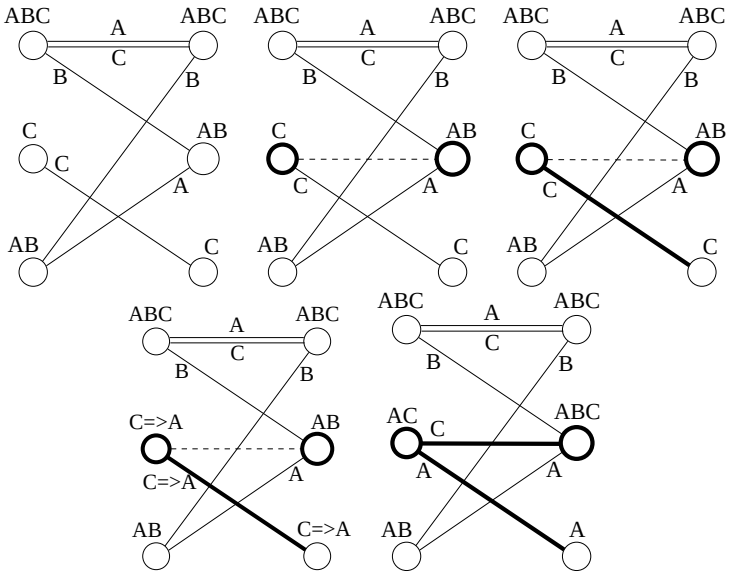

Fig. 7 An example for Swap.

This solution and the resulting C^{NL} and C^{RL} sets are shown in the top left part of figure 7.

A new request is the connection $o_7 = 4$ and $d_7 = 3$, which yields $k_7^1 = 1$ and $k_7^3 = 1$, see the middle top part of figure 7. The current solutions yields $C_{11}^A = C_1^{NL} \setminus C_1^{NR} = \{C\}$ and $C_{11}^B = C_1^{NR} \setminus C_1^{NL} = \{A, B\}$. We first try $c_A = C$ and $c_B = A$. This yields a one step path, (1,2), as can be seen in the top right part of figure 7.

Now we can change color C to color A for edge (1,2). This enables us to use color C for the new connection represented by edge (1,1). This is shown in the bottom parts of figure 7.

6 An alternate search strategy

A disadvantage of the method described above is that in order to find the shortest maximal path, we need to enumerate *all* possible pairs. If we are lucky, the number of possible pairs is small. However, if there are many possible pairs, a slightly different approach might be more efficient.

The idea here is first to look for a path of length one. If we find one, we are ready. If there is no path of length one, we search for a path of length two. If we find one, we are ready. Otherwise we search for path of length three, and so on.

Here it is important that a path needs to be maximal. A path is acceptable only when there is no possible continuation, which happens when a node has adjacent edges of only one of the two colors.

Recall that if $c_A \in C_i^{NL} \setminus C_j^{NR}$ and $c_B \in C_j^{NR} \setminus C_i^{NL}$, there exists an alternating path, starting at node i and with an edge of color c_A. Let k be the second node on this path. In order for this path to have length one, there should not be any edge of color c_B adjacent to node k. If this is the case, we may "swap" colors c_A and c_B along this

path, which in this case boils down to simply change k_m^2 from c_A to c_B. Then node i is freed from color c_A.

In short we wish to find a $c_A \in C_i^{NL} \setminus C_j^{NR}$. Then we let $k_1 = M^R(c_A, i)$, and wish to find a $c_B \notin C_i^{NL} \cup C_{k_1}^{NR}$. If we succeed, (i, k_1) is a one step path.

First we need to do as in Swap, namely to check if there is already a free color, or if the only possibility is to use a new color. If this has been done and not resulted in termination, we know that $c_B \in C_j^{NR}$ (since $c_B \notin C_i^{NL}$). Therefore we look for a $c_B \in C_j^{NR} \setminus (C_i^{NL} \cup C_{k_1}^{NR})$. In other words, we wish to know if $C_j^{NR} \setminus (C_i^{NL} \cup C_{k_1}^{NR}) = \emptyset$ or not. If it is not empty, any color, c_B, in this set can be used. That is, we require that $M^R(c_A, i) \geq 0$, $M^L(c_A, j) < 0$, $M^L(c_B, j) \geq 0$, $M^R(c_B, i) < 0$ and $M^L(c_B, k_1) < 0$.

Just as in Swap, we first use CreateSets(k_m^1, k_m^3) to get the sets $C_{k_m^1, k_m^3}^A$ and $C_{k_m^3, k_m^1}^B$.

Algorithm 1Path(i, j)

1. For each $c_A \in C_{ij}^A$
   (a) Let $k_1 = M^R(c_A, i)$.
   (b) For each $c_B \in C_{ji}^B$
      i. Let $l_1 = M^L(c_B, k_1)$.
      ii. If $l_1 < 0$, (i, k_1) is a one step path. Stop with success.

Note that the algorithm is terminated when we find the first path (since it is the shortest). There is at most d colors adjacent to a node, so the complexity of 1Path is $O(d^2)$. If this algorithm does not terminate with success, there is no maximal path of length one. In this case we have $C_j^{NR} \setminus (C_i^{NL} \cup C_{k_1}^{NR}) = \emptyset$.

For a path of length two, let l_1 be the third node. We need $c_A \in C_i^{NL} \setminus C_j^{NR}$, $c_B \in C_j^{NR} \setminus C_i^{NL}$, $c_A \in C_{k_1}^{NR}$, $c_B \in C_{k_1}^{NR}$, $k_1 = M^R(c_A, i)$, as well as $c_A \notin C_{l_1}^{NL}$, $c_B \in C_{l_1}^{NL}$, $l_1 = M^L(c_B, k_1)$. In other words, we require that $M^R(c_A, i) \geq 0$, $M^L(c_A, j) < 0$, $M^L(c_B, j) \geq 0$, $M^R(c_B, i) < 0$, $M^R(c_A, k_1) \geq 0$, $M^L(c_B, k_1) \geq 0$, $M^L(c_B, l_1) < 0$ and $M^R(c_A, l_1) < 0$.

We only do this if 1Path has failed, i.e. if we know that there exists no maximal path of length one. The algorithm will be as follows.

Algorithm 2Path(i, j)

1. For each $c_A \in C_{ij}^A$
   (a) Let $k_1 = M^R(c_A, i)$.
   (b) For each $c_B \in C_{ji}^B$
      i. Let $l_1 = M^L(c_B, k_1)$.
      ii. Let $k_2 = M^R(c_A, l_1)$.
      iii. If $k_2 < 0$, then $(i, k_1), (k_1, l_1)$ is a two step path. Stop with success.

As soon as we find a path of length two in 2Path, the algorithm is terminated. (Note that $l_1 < 0$ cannot happen in 2Path, since there is no one step path.) If 2Path does not stop with success, there is no maximal path with two edges.

The algorithm 2Path actually contains the same enumeration of colors as 1Path (although some more constant work is done in each iteration), so the complexity of 2Path is $O(d^2)$.

If both 1Path and 2Path fail, we use the following algorithm.

Algorithm 3Path(i, j)

1. For each $c_A \in C^A_{ij}$
   (a) Let $k_1 = M^R(c_A, i)$.
   (b) For each $c_B \in C^B_{ji}$
      i. Let $l_1 = M^L(c_B, k_1)$.
      ii. Let $k_2 = M^R(c_A, l_1)$.
      iii. Let $l_2 = M^L(c_B, k_2)$.
      iv. If $l_2 < 0$, then $(i, k_1), (l_1, k_1), (l_1, k_2)$ is a three step path. Stop with success.

Again, since the path is unique for a given pair of colors, the algorithm 3Path only enumerates the same colors as 1Path (and does some more constant work is done in each iteration), the complexity of 3Path is $O(d^2)$.

Let us generalize this. In order to find a maximal alternating path with K edges, we can use the following algorithm.

Algorithm KPath(i, j, K)

1. For each $c_A \in C^A_{ij}$
   (a) Let $k_1 = M^R(c_A, i)$.
   (b) For each $c_B \in C^B_{ji}$
      i. Set $q = 1$.
      ii. Let $l_q = M^L(c_B, k_q)$.
      iii. If $2q - 1 = K$: If $l_q < 0$, then $i - k_q$ is a K step path. Stop with success.
      iv. Let $k_{q+1} = M^R(c_A, l_q)$.
      v. If $2q = K$: If $k_{q+1} < 0$, then $i - l_q$ is a K step path. Stop with success.
      vi. If $2q \leq K$, set $q = q + 1$ and go to ii.

For a constant K the complexity is still $O(d^2)$. However, if K is allowed to grow, the complexity will be $O(Kd^2)$. We find that the stopping criteria are somewhat different for even and odd K. Note that KPath looks only for a path of length exactly K, and assumes that there is no shorter maximal path.

Let us now give the main algorithm, that first looks for a path of length one, then for a path of length two, and so on, until a path is found. It starts the same way as Swap.

Algorithm AnyPath(m)

1. If there exists a color such that $M^R(c_A, k^1_m) < 0$ and $M^L(c_A, k^3_m) < 0$, set $k^2_m = c_A$. Terminate.
2. Call CreateSets(k^1_m, k^3_m) in order to get $C^A_{k^1_m, k^3_m}$ and $C^B_{k^3_m, k^1_m}$. If any of these sets is empty, add a new color, c_C and set $k^2_m = c_C$. Terminate.
3. Set $K = 1$.
4. Call KPath(k^1_m, k^3_m, K). Terminate if success.
5. Set $K = K + 1$ and go to 4.

As a matter of practical efficiency, CreateSets is placed outside of the loop. The algorithm AnyPath cannot fail, since it continues until a path is found. A path cannot be longer then $2n - 2$, since it cannot contain a cycle or reach node k^3_m. Therefore, K never needs to be larger than $2n - 2$. The total complexity of AnyPath is therefore $O(n^2 d^2)$ (or $O(n^4)$).

This is worse than for the algorithm Swap, but in practice it may be faster if a fairly maximal short path exists, due to a better stopping criterion. The worst case complexity of AnyPath is actually very unlikely to occur in practice, since it assumes

that there are many colors than may be tried but all paths are still very long. In practice we believe that if there are many possible colors, the shortest path will be fairly short, while very long paths will only be obtained when there are few possible colors.

In our computational tests, we will observe both the lengths of the maximal paths and the number of pairs of colors that need to be checked.

7 Swapping more than two colors

One may wonder if it is enough to consider swaps between two colors. Since a two color swap cannot fail, we know that two color swaps are enough to find a new path. Therefore, the only remaining question is if a swap between three or more colors can produce a shorter path than two color swaps.

First we note that a two color swap only involves two switches, while a three color swap would involve three switches. So if the goal is to involve the minimal number of switches, two color swaps are obviously sufficient. Usually, however, the goal is a minimal number of rearrangements, i.e. paths with minimal number of links. Let us now consider swapping three colors.

Assume that the path starts with node i and edge (i,k) with color c_A. It then continues with edge (l,k) with color c_B and must contain at least one more edge, (l,m), with color c_C. A three color swap would then be to replace c_A with c_B for edge (i,k), replace c_B with c_C for edge (l,k), and replace c_C with c_A for edge (l,m).

From start we thus have $c_A \in C_i^{NL}$, $c_A \in C_k^{NR}$, $c_B \in C_k^{NR}$, $c_B \in C_l^{NL}$, $c_C \in C_l^{NL}$, $c_C \in C_m^{NR}$. In order for the swap to be possible, we must have $c_B \notin C_i^{NL}$, $c_C \notin C_k^{NR}$, $c_A \notin C_l^{NL}$, $c_A \notin C_m^{NR}$. Furthermore, this frees node i from color c_A, so $c_A \notin C_j^{NR}$ is needed for this to be useful. Since $c_B \notin C_i^{NL}$, we assume $c_B \in C_j^{NR}$, since otherwise we may use color c_B directly.

Summing up, we have a possible three color swap path, with three links, if the following holds: $M^R(c_A,i) = k$, $M^L(c_B,k) = l$, $M^R(c_C,l) = m$, $c_B \notin C_i^{NL}$, $c_C \notin C_k^{NR}$, $c_A \notin C_l^{NL}$, $c_A \notin C_m^{NR}$, $c_A \notin C_j^{NR}$, and $c_B \in C_j^{NR}$.

However, under these assumptions, we have a two color swap with only two links, namely the path i - k - l with $M^R(c_A,i) = k$ and $M^L(c_B,k) = l$. We can swap c_A and c_B along this path, since $c_B \notin C_i^{NL}$, $c_A \notin C_l^{NL}$. Since $c_A \notin C_j^{NR}$, this allows (i,j) to be colored by c_A. Note that we have not used the fact that $c_A \notin C_m^{NR}$, so the result is true even if the three color swap path is longer. The conclusion is the following.

Lemma 6 *If there exists a possible three color swap, there exists a two color swap with a shorter maximal path.*

Therefore it is sufficient to consider only two color swaps.

8 Implementation and computational tests

The algorithms have been implemented in C. Memory is required for a number of m-vectors (o, d, k^1, k^3, k^2) and two $n \times n$ matrices (M^R and M^L). We also found it convenient to use a $n \times n \times n$ structure, which yields l (the connection) from k^1, k^3 and k^2. This is used when finding the connections related to an alternating path. This

structure may be avoided, but then it will take a longer time to find the connection index.

We have tried the algorithms on randomly generated instances. An instance was generated as follows. First we let $o_l = l$ and $d_l = l$ for all l. Then we apply pairwise random swaps between elements in d. The vector d contains n^2 elements, and if we do n^4 swaps, the resulting d seems to be quite random in nature. (For larger problems, $n \geq 100$, we use fewer swaps, in order to limit the time needed to generate the instances.)

The real life application that initiated our interest in this problem has $n = 20$. However, it was fairly difficult to generate interesting and challenging test examples for such small instances. Routing one connection at a time, starting from scratch, most of the connections immediately found a free color, so swapping was only needed for a few connections. Furthermore, most maximal paths were fairly short.

Note that the two algorithms are identical when a free color is available. Differences only occur when swapping is needed.

One way of producing more interesting instances was to first route all connections that found free colors. This leaves a problem where swapping is needed for a majority of the remaining connections, and this can be used for testing. (Swapping is not needed for all remaining connections, as one might believe, since swapping may open possibilities of free colors for subsequent connections.)

We also generated larger instances, just in order to evaluate the algorithms, even if we do not know of real life instances that are that large.

First we give the computational results for solving the problems from scratch. Since most of the connections were made using free colors, a large proportion of the required solution time is identical in the two algorithms. In other words, the difference in solution time is small, relative to the total time. Therefore, even small differences are significant.

We generated 10 instances of each size, with n equal to 5, 10, 20, 100, 200, 300, 400 and 500. In table 2, we give the solution times in seconds, measured by the `clock`-function in C, not including time for reading problem data. The computer used is a 2.9 GHz PC with 512 MB RAM running Linux.

For all instances with $n \leq 20$, the time reported was zero, for both methods. (This means that the time is less than 0.01 seconds.) For all instances with $n = 100$, the time reported was 0.02 seconds for both methods. Therefore, in table 2, we only give the results for $n \geq 200$. The first number in the problem name is the number of nodes.

The method denoted by M1 is Swap, using Swap2 and CreateSets. The method denoted by M2 is AnyPath, using KPath and CreateSets. S0 denotes starting from scratch, while S1 means that we use the starting solution mentioned above.

We note that M2 in general is somewhat faster than M1, both for S0 and S1. In other words, it seems beneficial to avoid finding all paths, by searching for a path of a certain length at a time. The difference seems to grow as the size grows.

Concerning the numbers and lengths of paths, we give the results for one typical problem, namely 300-0. For this problem, we routed 90000 connections (in half a second). Of these, 88294 connections found a free color (i.e. could be routed without changing another connection), which means that 1706 connections (less than 2%) required swapping. Of these, 1516 connections could be routed with paths of length one (i.e. required the rerouting of only one connection). This means that only 190 connections required swapping paths of two or more edges.

Problem	M1S0	M2S0	M1S1	M2S1
200-0	0.12	0.12	0.00	0.02
200-1	0.10	0.12	0.02	0.00
200-2	0.12	0.12	0.00	0.00
200-3	0.10	0.12	0.02	0.02
200-4	0.12	0.10	0.02	0.02
200-5	0.12	0.12	0.00	0.02
200-6	0.14	0.12	0.00	0.00
200-7	0.14	0.12	0.02	0.02
200-8	0.12	0.12	0.00	0.00
200-9	0.12	0.12	0.02	0.02
300-0	0.46	0.44	0.04	0.04
300-1	0.46	0.44	0.04	0.02
300-2	0.48	0.46	0.04	0.02
300-3	0.48	0.42	0.04	0.04
300-4	0.46	0.44	0.04	0.02
300-5	0.54	0.46	0.04	0.02
300-6	0.48	0.42	0.04	0.04
300-7	0.46	0.42	0.04	0.02
300-8	0.48	0.44	0.06	0.04
300-9	0.46	0.42	0.04	0.02
400-0	1.20	1.14	0.10	0.08
400-1	1.20	1.12	0.10	0.06
400-2	1.22	1.12	0.10	0.08
400-3	1.18	1.14	0.10	0.08
400-4	1.24	1.22	0.10	0.06
400-5	1.24	1.16	0.08	0.08
400-6	1.18	1.10	0.10	0.06
400-7	1.18	1.12	0.10	0.08
400-8	1.20	1.12	0.12	0.08
400-9	1.18	1.12	0.10	0.08
500-0	4.82	4.16	0.52	0.70
500-1	4.28	4.58	0.62	0.50
500-2	4.68	4.58	0.58	0.58
500-3	4.56	4.38	0.56	0.54
500-4	4.36	4.30	0.74	0.56
500-5	4.40	4.28	0.68	0.54
500-6	4.56	4.28	0.62	0.52
500-7	4.48	4.36	0.84	0.78
500-8	4.42	4.24	0.82	0.54
500-9	4.56	4.08	0.60	0.54

Table 2 Computational results in seconds.

The maximal length of a path was 381 (i.e. routing this connection required the change of 381 other connections). The average length of a path (of those that needed swapping) was 2.2, so in average, if swapping is needed, a little more than two other connections need to be rerouted. Note however that this low number depends on the large number of paths of length one. The total average path length (also over the connections that did not need swapping) was 0.042. This is obviously due to the high number of connections that found a free color.

Concerning the number of possible pairs of colors, when swapping is needed, the maximal number was 5712, the minimal number was 1, and the average was 602. This is the number of times Swap calls Swap2. We thus find that this number is not very small, but much smaller than the worst case, $d^2 = 90000$.

Summing up, we find that there are some long paths, and also cases with fairly many pairs of colors, but these numbers are not at all as large as one might fear.

Starting from a solution with all connections not needing swaps already routed, we get the following for the same problem. Now only 1267 connections were routed, and of these 267 connections found a free color (due to previous swapping for other connections). Therefore 1000 connections needed swapping, and one step paths were found for 343 connections. The maximal length of a path was 348, and the average length was 7.70 over all routed connections, and 9.76 over those that needed swapping. The maximal number of pairs of colors was 308, the minimal was 1, and the average 28.9.

The only difference between the results of the tests with S0 and S1 is that much work that is identical in the two algorithms is removed. Since the reroutings change the conditions for subsequent connections, one should maybe see the tests with S1 simply as additional test problem instances.

Let us finally mention the possibility of first counting the number of possible pairs of colors, $|C^A_{k^1_m,k^3_m}||C^B_{k^3_m,k^1_m}|$, and then use Swap if this number is lower than a certain limit, and AnyPath if not. This might possibly utilize the advantages of both methods.

9 Conclusions

We study three stage symmetric Clos networks, and present graph optimization methods for finding the routing of new connection requests so that a minimal rerouting of already routed connections is done. The methods are guaranteed to give the exact optimum, have polynomial time complexity and are also efficient in practice.

The methods work by finding maximal alternating paths. One method is to find all maximal alternating paths and choose the shortest one, since it gives the minimal rerouting. The other one searches first for a path of length one. If none is found, it searches for a path of length two, and so on.

Computational tests indicate that the second approach is somewhat more efficient than the first one. However, solution times are fairly low for both methods.

A practical conclusion is that it is perfectly feasible to find the minimal rerouting solution in real time for many practical applications. The time needed for moderately sized instances is very small.

References

[1] Beneš, V.: *Mathematical Theory of Connecting Networks and Telephone Traffic.* Academic press. (1965)
[2] Berge, C.: *Graphs and Hypergraphs.* North-Holland, Amsterdam. (1973)
[3] Carpinelli, J. D., and Oruç, A. Y.: Applications of matching and edge-coloring algorithms to routing in Clos networks. *Networks* **24**, 319–326. (1994)
[4] Clos, C.: A study of nonblocking switching networks. *Bell Syst. Tech. J.* **32**, 406–424. (1953)
[5] Cole, R., and Hopcroft, J.: On edge coloring bipartite graphs. *SIAM Journal on Computing* **11**, 540–546. (1982)
[6] Cole, R., Ost, K., and Schirra, S.: Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica* **21**, 5–12. (2001)
[7] Holmberg, K.: Optimization models for switching networks of Clos type with many stages. *AMO - Advanced Modeling and Optimization* **10:1** (2008)

[8] Hwang, F. K.: Control algoritmhs for rearrangeable Clos networks. *IEEE Transactions on Communications* **31**, 952–954. (1983)
[9] Lee, H. Y., Hwang, F. K., and Carpinelli, J. D.: A new decomposition algorithm for rearrangeable Clos interconnection networks. *IEEE Transactions on Communications* **44**, 1572–1578. (1996)
[11] Schrijver, A.: Bipartite edge-coloring in $O(\Delta m)$ time. *SIAM Journal on Computing* **28**, 841–846. (1999)