A Parallel Route Assignment Algorithm for Fault-Tolerant Clos Networks in OTN Switches

Lingkang Wang, Tong Ye, *Member, IEEE*, and Tony T. Lee, *Fellow, IEEE*

Abstract—The three-stage fault-tolerant Clos network, where extra switch modules exist in the middle stage in case of switch failures, is widely used in the design of OTN switches. This paper proposes a route assignment algorithm for such Clos networks by solving its counterpart in edge-coloring problem. Based on complex coloring, a novel edge coloring method, the proposed algorithm possesses two properties. First, our algorithm can make full use of extra switch modules in the middle stage of Clos network. The extra switch modules provide additional colors for edge coloring, which help to reduce the running time of the coloring process remarkably. Second, our algorithm can be implemented in a parallel manner to further shorten the running time. The proposed routing algorithm achieves a low complexity of $O(\frac{\sqrt{N}(m-1)}{m-1+(m-\sqrt{N})\log N}\log N)$, where N is the network size and m is the number of switch modules in the middle stage. The performance of our algorithm has been verified by extensive simulation experiments.

Index Terms—Route assignment, fault tolerance, Clos network, optical switching, edge coloring, complex coloring

1 Introduction

With the explosive growth of Internet traffic, large-scale optical switches become more important [1], [2], [3]. This is especially true since optical transport network (OTN) has recently been considered as the mainstream solution for the wireless front-haul system of 4G/5G networks [4], [5], [6], [7], [8]. Currently, even for a medium-size urban network, an optical switch in a centralized baseband pool is required to switch up to 1,000 macro base station carriers [9], each of which will provide the bandwidth of 100 Gbps, according to 5G requirements [8]. Thus, a single optical switch has to provide throughput greater than 100 Tbps, when the OTN is used to support 4G/5G networks.

Because they feature good scalability and internally non-blocking property, Clos networks have been widely adopted in the construction of large-scale optical switches in OTNs [10], [11], [12], [13], [14], [15], [16]. For example, Cisco offers a converged optical service platform, NCS 4,016 [14], in which the switching modules are organized in a Clos-type configuration. Also, Huawei recently demonstrated a prototype of Clos networks for the OTN multi-chassis switch cluster [15], which can support 1 Pbps switching capacity and promises a bright future for OTN switch applications.

The Clos network is a three-stage switching network consisting of a set of switching modules. In a symmetric Clos network, denoted by $C(m,n,r)$, there are r $n\times m$ input modules (IMs) in the input stage, m $r\times r$ central modules (CMs) in the middle stage, and r $m\times n$ outputs modules (OMs) in the output stage. Fig. 1 gives an example of $C(m,n,r)$. There is exactly one link between two switching modules at two adjacent switching stages. Thus, the middle stage essentially provides m alternative paths for each pair of IM and OM, and each CM corresponds to one such path. The non-blocking routing problem in Clos networks is to assign CMs to the connection requests between the IMs and the OMs, such that there is no contention: two connections share the same IM or OM do not pass through the same CM. It is well-known that the Clos network is rearrangeably non-blocking (RNB) if and only if there are at least n available paths for each IM-OM pair, i.e., $m \geq n$.

When some optical switching modules break down, due to device aging or overloading operations [16], [17], [18], [19], the failure may disconnect multiple connections, leading to the loss of a large amount of data. For example, each central module of the OTN switch proposed in [15] is a 324×324 optical switching module with a line rate of 12.5 Gbps. If such a module fails, the degradation of throughput is up to terabits. Therefore, fault tolerance is vital to supply relief for Clos networks if a failure occurs in practice [16], [18], [19], [20], [21].

The module failures may occur in different switching stages. If failures happen in an IM or OM, the input or output ports of the affected connections will be completely disconnected from the switching network. The only way to recover the connections is to replace the broken switching module with a functional one. However, if a CM fails in a Clos network $C(m,n,r)$, where the number of CMs m in the middle stage is greater than n, the Clos network $C(m,n,r)$ is still RNB, and can recover the interrupted communications by reallocating other CMs to those affected connections. In this paper, we refer to the Clos network $C(m,n,r)$

- *L. Wang and T. Ye are with the State Key Laboratory of Advanced Optical Communication Systems and Networks, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: {sin7cera, yetong}@sjtu.edu.cn.*
- *T.T. Lee is with the Chinese University of Hong Kong (Shenzhen), Shenzhen 518172, China. E-mail: tonylee@cuhk.edu.cn.*

Manuscript received 28 Apr. 2018, revised 10 Oct. 2018, accepted 5 Nov. 2018, Date of publication 12 Nov. 2018; date of current version 10 Apr. 2019. (Corresponding author: Tong Ye.)
Recommended for acceptance by P. Balaji.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2880782

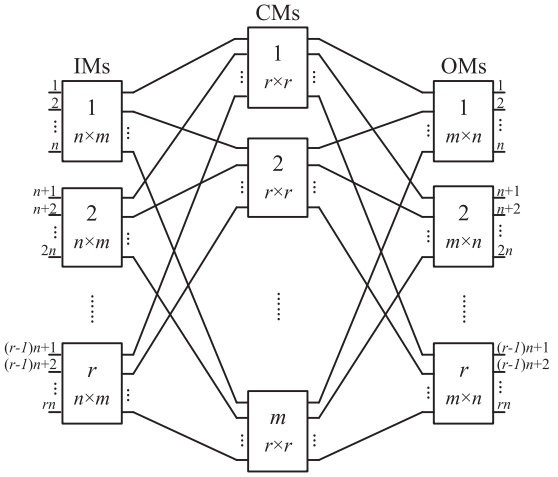


Fig. 1. A three-stage Clos network architecture $C(m, n, r)$.

with $m > n$ as a fault-tolerant Clos network, and focus on how to cope with the CM failures.

There are two methods to configure extra CMs in the middle stage. In the first method [22], a set of $m - n$ CMs is idle in the normal state. When a failure happens, the network directly shifts the affected connections to one of these $m - n$ idle CMs. This method is easy to implement and requires no additional computation cost to re-route the affected connections upon failure. However, this method does not take full advantages of idle CMs, which may help reduce the complexity of route assignment. Intuitively, the more CMs in the middle stage, the more alternative paths the Clos network can use for each input-output pair, and thus the more flexibility may be introduced to the route assignment [21]. Therefore, a second method was considered in previous works [19], [20], [21], where the m CMs were all employed during the route assignment in the normal state. However, most of the existing fault-tolerant routing algorithms [19], [20], [21] still cannot make full use of such routing flexibility introduced by the extra CMs.

1.1 Overview of Previous Works

The most efficient sequential route assignment algorithm of a Clos network is the one reported in [23], of which the time complexity is $O(nr \log m)$. To reduce the time complexity, several parallel algorithms were proposed in [24], [25], [26]. For example, parallel algorithms proposed in [25], [26] carry out conflict-free routing based on the concept of equivalence class. The time complexity of such parallel algorithms is $O(\log m \times \log nr)$ [26]. It is clear that the time complexity of the above route assignment algorithms increases with the number of CMs m, which means the classical routing algorithms cannot take full advantage of the flexibility provided by extra CMs. In general, directly applying classical route assignment algorithms [23], [24], [25], [26] to the fault-tolerant Clos networks is not desirable. The following paragraphs briefly describe several routing algorithms dedicated to fault-tolerant Clos networks.

The algorithms reported in [19], [20] deal with the route assignment problem when there are cross-point failures in some CMs. A normal switching module can be configured to an arbitrary switching state. When there are cross-point failures in the module, the number of switching states that the module can achieve is limited. The algorithm in [19] first calculates a set of n switching states that the middle stage should provide according to the connection requests, and then associates each switching state with a CM if this CM can implement this switching state. The time complexity of this routing algorithm is $O(|LC|\log nr + (nr)^{1.5})$, where $|LC|$ is the number of broken cross-points in the Clos network.

Another routing algorithm, called FT-DSRR, was developed in [20]. In this algorithm, each IM independently seeks the CMs that can be used to establish connections for the requests that originate from this IM. This algorithm is blocking, and not all legitimate requests can be satisfied at the same time. Thus, the algorithm requires buffers installed in the CMs and the OMs to avoid contention. In addition, the time complexity of this algorithm is $O(nm)$, which also increases with the number of CMs m.

Ref. [21] considered the case, where the CM will be treated as an unusable module even when there is only one broken cross-point. In ref. [21], the routing algorithm was devised based on a specification matrix, in which each row and each column respectively represent an IM and a CM, and each entry records the index of the OM that the IM can reach via the CM. The routing constraint requires that each OM appears exactly once in each column if the CM assignment is contention-free. Initially, a matrix is randomly generated, and the algorithm swaps the entries in the same row iteratively until the matrix is contention-free. Though this paper points out that extra CMs can reduce the number of required swaps and thus shorten the running time of the algorithm, it only analyzes the time complexity of this algorithm when there is no extra CMs, which is $O(nr^2)$.

In summary, the existing algorithms did not fully utilize the routing flexibility provided by the extra CMs. There is still room to improve the complexity of route assignment algorithms for optical switches that are based on three-stage fault-tolerant Clos networks.

1.2 Summary of our Work

In this paper, we propose a parallel routing algorithm for fault-tolerant Clos networks that fully utilize the routing flexibility of redundant optical switch modules to achieve a low time complexity. The routing problem in a fault-tolerant Clos network is formulated as an edge coloring of a bipartite graph with redundant colors. We apply a newly proposed edge-coloring method [27], called complex coloring, to solve the coloring problem. As a kind of algebraic method, we show that complex coloring possesses two attractive features as follows. First, complex coloring makes full use of redundant colors to shorten the running time of the coloring process. Our analysis demonstrates that the greater the number of redundant colors, the faster the coloring process. Second, complex coloring is very fast since it can be implemented in a parallel fashion.

Based on parallel complex coloring, the time complexity of the proposed routing algorithm is on the order of $O(\frac{n(m-1)}{m-1+(m-n)\log r}\log r)$ for Clos network $C(m, n, r)$. When we assume $n = r$ and $N = nr$ is the network size, the time complexity of our algorithm is expressed as $O(\frac{\sqrt{N}(m-1)}{m-1+(m-\sqrt{N})\log N}\log N)$. This indicates that our algorithm can make full use of redundant optical switches to improve the running speed. Also, such low complexity

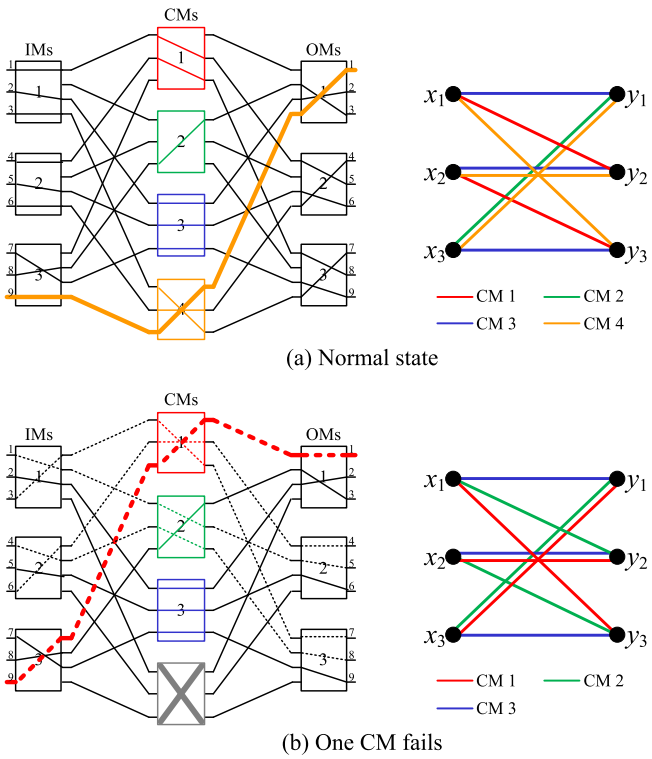


Fig. 2. Correspondence between the route assignment in a fault-tolerant Clos network and the edge coloring of a bipartite graph.

guarantee that the network can immediately recover all interrupted connections when switch failures occur.

The rest of the paper is organized as follows. In Section 2, we formulate the route assignment problem in the fault-tolerant Clos network as an edge-coloring problem of bipartite graphs with redundant colors. Next, we briefly describe complex coloring and its properties. In Section 3, we devise a parallel algorithm of complex coloring, and demonstrate how to accelerate the parallel complex coloring process by redundant colors. In Section 4, we propose a route assignment algorithm for fault-tolerant Clos networks, and show that our algorithm remarkably outperforms the previous algorithms in terms of time complexity. Furthermore, we also address the scalability issues of our parallel algorithm in this section. Section 5 draws the conclusion of this paper.

2 Route Assignment and Parallel Complex Coloring

The route assignment in fault-tolerant Clos networks can be formulated as the edge coloring of bipartite graphs [28], [29], [30]. In this section, to build this equivalence, we first briefly introduce the route assignment in the fault-tolerant Clos network. Then, we describe a newly proposed edge coloring method [27], called complex coloring, and its parallel processing in bipartite graphs [31].

2.1 Route Assignment in Fault-Tolerant Clos Network

In this paper, we consider a symmetric fault-tolerant Clos network $C(m, n, r)$, where $m > n$. Such Clos networks can remain RNB even when up to $m - n$ CMs are broken. An example of the fault-tolerant Clos network is illustrated in

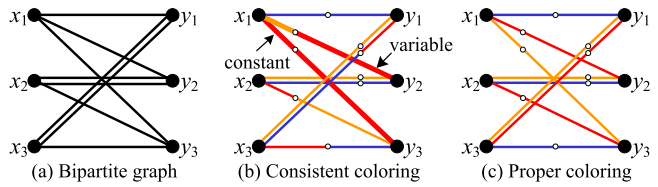


Fig. 3. Complex coloring of a bipartite graph G.

Fig. 2a, where $n = r = 3$ and $m = 4$. In this example, the network is fully loaded, since the inputs are all busy. If the fourth CM fails, three connections will be disconnected. However, the Clos network can recover the affected connections, since the other three CMs are still in normal state. For example, the connection passing through CM 4 in Fig. 2a is re-routed via CM 1 in Fig. 2b when CM 4 is broken.

The route assignment algorithm is the key to route (or re-route) the connections in fault-tolerant Clos networks [28], [29], [30]. As Fig. 2 illustrates, the route assignment of a fault-tolerant Clos network can be formulated as the edge coloring of a bipartite multigraph $G = (X \cup Y, E)$, in which the vertex sets X and Y represent the IM set and the OM set, respectively, and each edge in E represents a connection request from an IM to an OM. Let C be the color set, where each color in C corresponds to a CM in $C(m, n, r)$. Contention-free routing in Clos networks requires that any two connections originated from the same IM or destined for the same OM should not pass through the same CM, which is consistent with the constraint of edge coloring in G that two edges incident to the same vertex should use different colors.

Let Δ be the maximum degree of G. It is well-known that a bipartite graph G is Δ-edge-colorable [32], i.e., the minimum number of colors required to color the bipartite graph G is Δ. In the Clos network $C(m, n, r)$, there are m CMs and each IM (or OM) has n inputs (or outputs), which implies C corresponds to a bipartite graph G with $\Delta = n$ and $|C| = m$. Since G is n-edge-colorable, n CMs in the middle stage are sufficient to guarantee that the Clos network $C(m, n, r)$ is RNB. In a fault-tolerant Clos network, the number of CMs m is larger than n, which means the corresponding graph G has redundant colors for edge coloring. As Fig. 2a illustrates, G has $|C| = 4$ colors available for edge coloring, even though $\Delta = 3$. The graph G with $|C| > \Delta$ is referred to as a bipartite graph with redundant colors; otherwise, G is a bipartite graph without redundant colors. We show in Section 3 that these redundant colors will introduce flexibility in edge coloring.

2.2 Complex Coloring of Bipartite Graph

The complex coloring proposed in [27] is a kind of algebraic edge coloring method, in which a fictitious vertex is inserted in the middle of each edge such that the edge is divided into two links. As an example, the bipartite graph in Fig. 3a changes to the graph in Fig. 3b after a fictitious vertex is inserted in the middle of each edge.

The complex coloring starts coloring the links instead of the edges. Each vertex randomly selects a color for each link such that the links incident to this vertex use different colors, which is called consistent. It is clear that G can be consistently colored by Δ colors, where Δ is the maximum degree of the graph. An example of consistent coloring of Fig. 3a is given in Fig. 3b. In a consistently colored graph, the two links

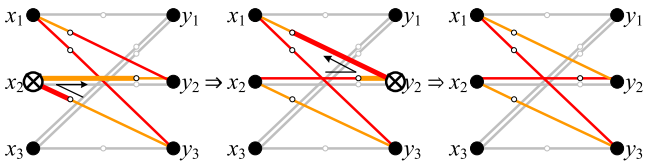


Fig. 4. Variable elimination via color-exchange operations.

of each edge may be colored by two different colors. If the colors of two links are different, the associated edge is called a variable and denoted as (a, b), otherwise it is called a constant, as Fig. 3b shows. A coloring of a graph is proper if it is consistent and all edges are constant. Fig. 3c illustrate a proper coloring of the bipartite graph displayed in Fig. 3a.

A proper coloring of G can be obtained from any initial consistent coloring by variable elimination. The variable elimination is achieved by color-exchange operations, which exchange colors of two links incident to the same vertex. An example is illustrated in Fig. 4, where a sequence of color exchanges is performed at the vertexes marked by "$\otimes$". In particular, after two exchanges, a (r, o) variable hit another (r, o) variable and is eliminated. The color exchange is effective if it does not increase the number of variables. In complex coloring, only effective color exchanges are allowed, such that the increase of the number of variables can be prevented. For example, the color exchanges are all effective in Fig. 4.

In bipartite graphs, all the variables can be eliminated via color-exchange operations, since there is no odd cycle [27]. Furthermore, complex coloring of bipartite graphs possesses the following three powerful strengths [31]:

1) Invariance of color set: The color-exchange operation does not introduce new colors to the current coloring, and thus the color set stays the same from an initial consistent coloring configuration to a final proper coloring configuration.
2) Rearrangeability: When new connection requests arrive or some existing connections needs to be rerouted due to the failure of optical central modules, it is only necessary to eliminate the new variables introduced by the new requests, instead of recoloring the entire bipartite graph.
3) Parallelizability: In a bipartite graph G, vertices in set X are non-adjacent, and so are vertices in set Y. Thus, color exchanges can be effective even when they are simultaneously performed on X, and on Y alternately. An example of parallel color-exchange operations is plotted in Fig. 5. In the first iteration, the vertices x_1, x_2, and x_3 execute color exchanges in parallel, and then y_3 performs color exchanges in the second iteration. Clearly, this property can remarkably improve the efficiency of the edge coloring process, since multiple variables may be eliminated at the same time [31].

Though the parallelizability property can speed up the variable elimination process, such a parallel process may introduce deadlock variables that are trapped in infinite loops and can never be eliminated [31]. For example, the path passing through x_1, y_2, x_2 and y_3 in Fig. 5 forms a (r, y) cycle containing two (r, y) variables. These two variables move forward in the same direction in a cycle and thus

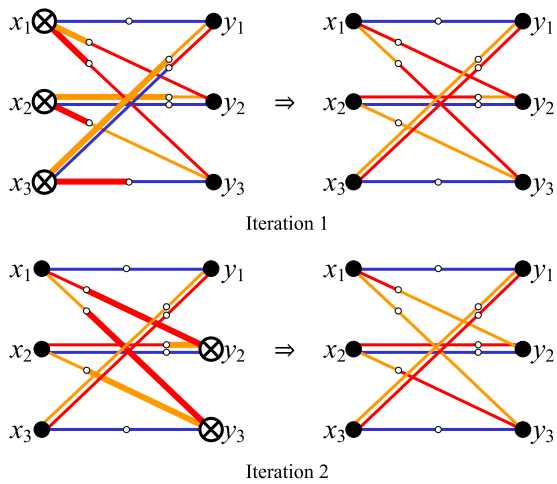


Fig. 5. Parallel processing of color-exchange operations.

have synchronous walks, causing an infinite loop. Such deadlocks are inherently attributed to the fact that each vertex lacks the global information about the entire graph. Fortunately, simulation results in [31] show that the number of remaining variables in G is rare if the parallel processing of color-exchange operation is executed after $O(\log N)$ iterations. For this fact, a stopping rule is introduced in [31] to halt the parallel variable elimination process after $O(\log N)$ iterations, such that the number of remaining variables can be smaller than a preset threshold.

3 Parallel Complex Coloring with Redundant Colors

In the bipartite graph counterpart of fault-tolerant Clos networks, the number of available colors is greater than its maximum degrees. Specifically, each vertex is connected to at most Δ edges but have $\Delta + \delta$ available colors. In Section 3.1, we show that such redundant colors essentially provide a new way for variable elimination. In Section 3.2, we further elaborate that such redundant colors can remarkably shorten the parallel variable elimination process such that the number of remaining variables can decline to the preset threshold much faster, which yields a parallel complex coloring algorithm with redundant colors that Section 3.3 describes.

3.1 New Features of Variable Elimination with Redundant Colors

In a bipartite graph G with $\Delta + \delta$ colors, the redundant colors essentially provide a new way to eliminate variables. It is clear that each vertex always has δ unused colors, since there are at most Δ links attached to this vertex. As a result, the walk of a variable may terminate on a vertex where no link of the required color exists to exchange. Fig. 6 illustrates an example, where the (r, b) variable hits vertex x_3 without b colored link. In this case, the (r, b) variable can be eliminated by directly replacing the color r with the unused color b. The (r, b) variable in Fig. 6 is said to be eliminated via the color-exchange operation with a *don't care edge*.

In a bipartite graph with redundant colors, an (a, b) variable is eliminated by either an $(a, *)$ variable or a *don't care edge*. These two kinds of elimination events are mutually exclusive. For an (a, b) variable, the condition of hitting a

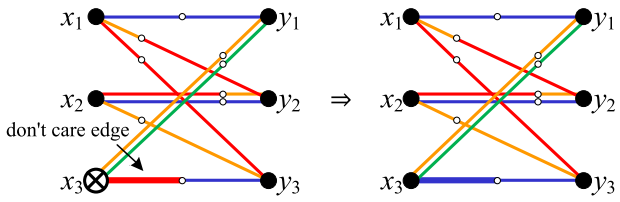


Fig. 6. Variable elimination via color-exchange operations with a *don't care edge*.

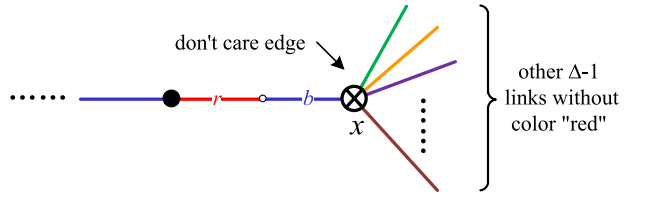


Fig. 7. Illustration of the case where an (r, b) variable hits a *don't care edge*.

don't care edge on a vertex is that color a is unused by the links attached to this vertex, while that of hitting an $(a, *)$ variable is that color a has been already used by the links attached to this vertex. For example, the (r, o) variable between x_2 and y_2 in Fig. 6 meets another (r, o) variable at y_2. In this case, it's impossible for the (r, o) variable to meet a *don't care edge* at y_2 at the same time, since the color red is used by one link incident to y_2.

Also, the features of these two kinds of variable eliminations are quite different. The number of variables declines with evolution of the variable elimination process, while the number of *don't care edges* does not decrease. The (r, b) variable in Fig. 6 eliminated at vertex x_3 is an example. This variable is eliminated by the replacement of color r with color b at x_3. However, this *don't care edge* still exists when another (r, b) variable reaches x_3, in which case x_3 is a vertex without r colored link.

Therefore, *don't care edges* provide a good complementation during the complete variable elimination process. To elaborate this point, we perform an ideal analysis. In particular, we consider the following two parameters:

1) $\alpha(t)$: *Variable elimination rate*, the ratio of the number of eliminated variables in the tth iteration to the total number of variables in this iteration.
2) $h(t)$: *Hitting time*, the expected number of iterations needed for a variable to hit another variable or *don't care edge* in the tth iteration. Obviously, $h(t)$ is inversely proportional to $\alpha(t)$.

During the analysis, we consider the following two ideal assumptions:

A1: $\Delta + \delta$ colors are randomly assigned to links during the graph initialization, and
A2: elimination rate $\alpha(t)$ is a constant α with respect to t.

Since a variable is either eliminated by another variable or a *don't care edge*, the variable elimination rate consists of the following two mutually exclusive parts:

P1: Probability that the variable is eliminated by a variable;
P2: Probability that the variable is eliminated by a *don't care edge*.

For the first part, the derivation in [31] shows that the probability of a variable being eliminated by hitting another variable in each iteration is $\frac{1}{a\log(|V|+b)+c}$, where a, b, c are constant. An intuitive explanation is that the average path length in a random graph is on the order of $O(\log|V|)$ [33], which implies that the average hitting time for a variable to meet another variable is $O(\log|V|)$.

For the second part, we consider an (r, b) variable incident to a vertex $x \in X$ with degree Δ in Fig. 7 and the color-exchange operation is currently performed at this vertex. For the vertex x, color b has been used, which means other $\Delta - 1$ links are colored by $\Delta - 1$ colors randomly selected from a set of $\Delta + \delta - 1$ colors. Hence, the total number of possible color assignments is $\binom{\Delta+\delta-1}{\Delta-1}$. If this variable hits a *don't care edge*, color r must be unused by the links attached to vertex x, as Fig. 7 shows. In this case, other $\Delta - 1$ links incident to x are colored by $\Delta - 1$ colors randomly selected from a set of $\Delta + \delta - 2$ colors without colors r and b. In other words, only $\binom{\Delta+\delta-2}{\Delta-1}$ color assignments are possible. It follows that the probability that a variable hits a *don't care edge* in each iteration is given by

$$p = \Pr\{\textit{hitting a don't care edge}\} = \frac{\binom{\Delta+\delta-2}{\Delta-1}}{\binom{\Delta+\delta-1}{\Delta-1}} = \frac{\delta}{\Delta+\delta-1}. \quad (1)$$

Equation (1) clearly indicates that probability p increases with δ, which is attributed to the fact that more redundant colors lead to more *don't care edges* and thus provide more options for the variable elimination.

Since probabilities P1 and P2 are mutually exclusive, the elimination rate α is given by

$$\alpha = \frac{1}{a\log(|V|+b)+c} + p, \quad (2)$$

which yields the hitting time

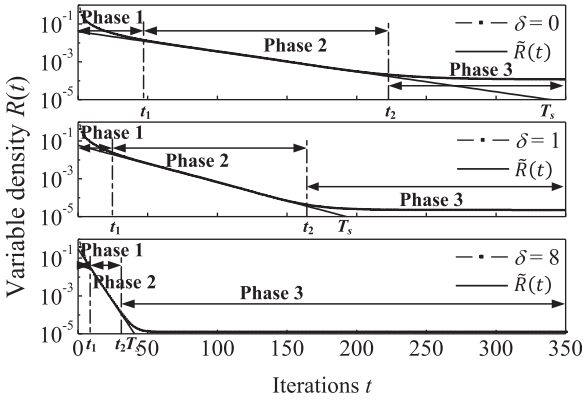
$$h = \frac{1}{\alpha} = \frac{1}{\frac{1}{a\log(|V|+b)+c} + \frac{\delta}{\Delta+\delta-1}} = \frac{1}{1+(1-\frac{\Delta-1}{\Delta+\delta-1})[a\log(|V|+b)+c]}[a\log(|V|+b)+c]. \quad (3)$$

From (2) and (3), its easy to see that the hitting time h reduces drastically with the increase of the number of redundant colors δ. When $\delta = 0$, i.e., there is no redundant colors in the graph, hitting time h is the maximum, and thus the elimination rate α is the smallest. In this case, hitting time h degenerate into $O(\log|V|)$, the same with that in [31], which considers the variable elimination in a bipartite graph G with Δ colors.

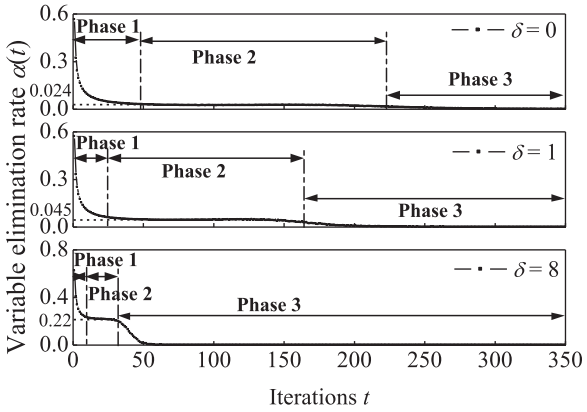
3.2 Accelerated Parallel Elimination Process

Because of the increase of the variable elimination rate, the *don't care edges* can accelerate the parallel variable elimination process. To demonstrate this point, we study the ratio of the number of variables after t iterations to the number of edges $|E|$, which is called variable density and denoted as $R(t)$.

According to our previous work [31], under the ideal assumptions A1 and A2, variable density $R(t)$ is given as follows:



(a)



(b)

Fig. 8. Three phases of parallel complex coloring of a bipartite graph with different δ, where $|V| = 128$ and $\Delta = 32$.

$$R(t) = (1-\alpha)^t R(0), \tag{4}$$

For a bipartite graph with redundant colors, α is given by (2). Thus, for $\alpha \ll 1$, from (2) and (4), the expected number of iterations T to achieve a given density ϵ is obtained as follows:

$$\begin{aligned} T &\approx \frac{\ln \epsilon - \ln R(0)}{-\alpha} \\ &= \frac{(\Delta - 1 + \delta)[a \log(|V| + b) + c]}{\Delta - 1 + \delta[1 + a \log(|V| + b) + c]} \ln \frac{R(0)}{\epsilon}, \end{aligned} \tag{5}$$

where a, b, and c are constant. This result implies that, under the ideal assumptions, the parallel complex coloring with δ redundant colors requires $O(\frac{(\Delta-1+\delta)[a\log(|V|+b)+c]}{\Delta-1+\delta[1+a\log(|V|+b)+c]})$ iterations to achieve a specific variable density ϵ. It is clear that the redundant colors drastically accelerate the variable elimination process.

In reality, however, the ideal assumption A2 is not always satisfied. We carry out extensive simulations under three different values of δ, namely 1, 2, and 8. For each δ, 100,000 bipartite graphs with $|V| = 128$ and $\Delta = 32$ are randomly generated. The simulation results in Fig. 8 show that all the variable elimination processes with different δs experience three phases, according to the behavior of variable density $R(t)$ and elimination rate $\alpha(t)$:

1) Initial phase: $R'(t) < 0$, $\alpha'(t) < 0$. In this phase, the initial variable density is very high and the variables are more likely to be eliminated. Thus, the rate $\alpha(t)$ is large and the density $R(t)$ decreases dramatically.
2) Steady phase: $R'(t) < 0$, $\alpha'(t) = 0$. In this phase, variables are scattered over the graph and the elimination enters a relatively stable phase. The rate $\alpha(t)$ is stabilized and becomes a constant, and the density $R(t)$ drops off with the same slope. As Fig. 8 shows, this phase is the principal part of the whole process.
3) Deadlock phase: $R'(t) = 0$, $\alpha'(t) = 0$. In this phase, the density $R(t)$ remains unchanged.

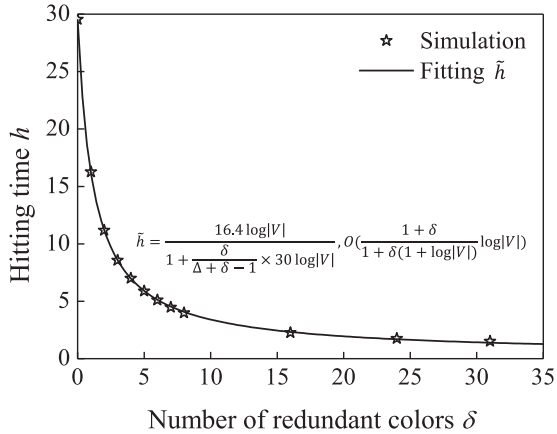


Fig. 9. Hitting time h as a function of the number of redundant colors δ, where $|V| = 128, \Delta = 32$.

Fig. 8 clearly indicates that the redundant colors can substantially speed up the elimination process in practical situations. In the following, we take the steady phase of the elimination process as an example to elaborate this point. When δ is large, the number of *don't care edges* is large. Also, as Section 3.1 explains, the number of *don't care edges* does not decrease, though a lot of variables have been eliminated in the initial phase. These *don't care edges* help the elimination probability $\alpha(t)$ to stabilize at a higher value in the steady phase. For example, $\alpha(t)$ in Fig. 8b increases from 0.024 to 0.22 when δ increases from 0 to 8. It follows that the variable density $R(t)$ declines faster in the steady phase and the duration time of the steady phase becomes shorter with the increase of δ.

To further demonstrate the advantage of redundant colors, we plot the hitting time h of the steady phase changing as a function of δ in the Fig. 9. It's clear to see that h dramatically decreases with the increase of δ, and the curve matches with (3) very well.

The parallelizability inherently results in deadlock, as we explain in Section 2. We thus introduce a stopping rule to halt the endless parallel process. For the choice of the stopping rule, we only consider the steady phase since the initial phase is much shorter than the steady state and the deadlock phase will be treated in a different way, which will be discussed in Section 3.3. In the steady phase, the rate $\alpha(t)$ is almost a constant, and the density $R(t)$ in the steady phase can be given according to (4) as follows:

$$R(t) = (1-\alpha)^{t-t_1} R(t_1), \tag{6}$$

where t_1 and t_2 are respectively the start and end time of the steady phase, and $t \in [t_1, t_2]$. For a given variable density $\epsilon \ll 1$, the required number of iterations T_s is given by

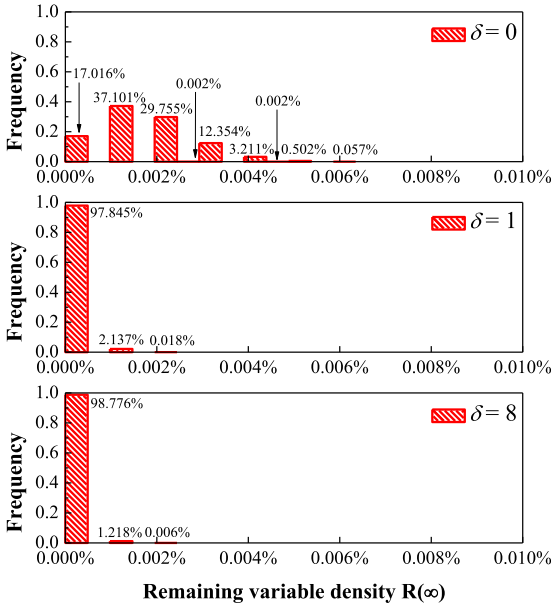


Fig. 10. Distribution of remaining variable density, where $|V| = 128$ and $\Delta = 32$.

$$T_s = \frac{1}{\alpha}\ln\frac{R(t_1)}{\epsilon} + t_1, \\ = \frac{(\Delta - 1 + \delta)R(t_1)[a\log(|V| + b) + c]}{\Delta - 1 + \delta[1 + a\log(|V| + b) + c]} + t_1, \tag{7}$$

where a, b, and c are constant. According to (6), we plot the dashed line, denoted as $\tilde{R}(t)$, in Fig. 8a. The cross-point of $\tilde{R}(t)$ and $R(t) = \epsilon$ gives T_s. For example, the stopping time T_s is marked for $= 10^{-5}$ in Fig. 8a. As we can see, the elimination process has already entered the deadlock phase at the stopping time T_s, which indicates that our selection of stopping time is conservative for this example.

Though few variables may be left after the stopping time T_s, *don't care edges* can also remarkably reduce the number of variables entering the deadlock phase. This point is confirmed by simulation experiments in Fig. 10, of which the simulation condition is the same with that of Fig. 8. Fig. 10 plots the distribution of the density of the leftover variables when the parallel color exchanges are iterated T_s times. The simulation results show that, the density of leftover variables $R(T_s)$ dramatically decreases, even when the number of redundant colors δ increases from 0 to 1. For example, only 17.016 percent experiments for $\delta = 0$ do not have deadlock variables, while the ratio increases to 97.845 and 98.776 percent, respectively when $\delta = 1$ and $\delta = 8$. This indicates the parallel elimination process almost does not introduce deadlocked variables with the presence of redundant colors.

3.3 Parallel Coloring Algorithm with Redundant Color

Based on the above discussion, we propose a parallel coloring algorithm for a bipartite graph $G = (X \cup Y, E)$, of which the maximum degree is Δ and the number of available colors is $\Delta + \delta$. The algorithm starts with a consistent coloring of graph G using $\Delta + \delta$ colors. During the variable elimination process, color exchanges are alternately performed on vertices in set X and then on vertices in set Y. This parallel procedure is iterated until all variables are eliminated, or terminated subject to the stopping time. If there are leftover variables due to deadlock situations, the algorithm eliminates them one-by-one by sequential complex coloring. Since no odd cycle exists in bipartite graphs, the remaining variables can all be eliminated and thus a proper coloring can be obtained [27]. Fig. 11 illustrates the sequential variable elimination of the deadlock situation in Fig. 5. In this example, all deadlock variables are finally eliminated and a proper coloring is obtained.

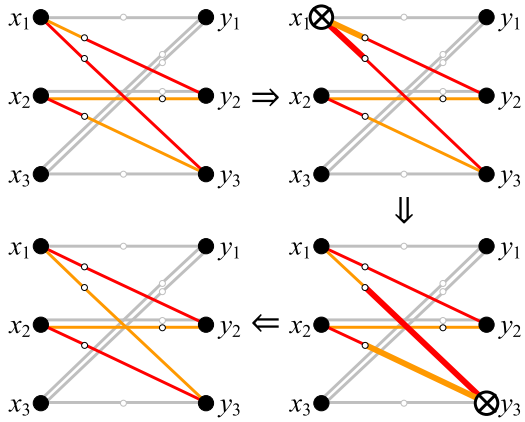


Fig. 11. Sequential elimination process of remaining variables.

We now present the algorithm *Parallel Complex Coloring with Redundant Colors* as Algorithm 1.

Algorithm 1. Parallel Complex Coloring with Redundant Colors

Input: A bipartite graph $G = (X \cup Y, E)$, maximum degree Δ, color set C with $|C| = \Delta + \delta, \delta > 0$, $L(x_i)$ (or $L(y_i)$): the list of variables incident to vertex x_i (or y_i), stopping time T_s.

Output: A properly colored graph G.

1: For each vertex in G, choose a random color out of $\Delta + \delta$ colors in C for each of its associated links.

2: For each vertex x_i in X, update $L(x_i)$ and proceed with variable elimination process in parallel: If $L(x_i)$ is nonempty, then for each (a, b) variable in $L(x_i)$, one of the following operations is executed:

- O1: If there is an adjacent link with color a, find it and do the color-exchange operation.
- O2: If there is no adjacent link with color a, replace (a, b) variable with (a, a) constant directly.

3: For each vertex y_i in Y, do the same procedure as Step 2.

4: Repeat Steps 2-3, until one of the following two conditions is satisfied:

- C1: There is no variable left in graph G.
- C2: The predetermined stopping time T_s is up.

5: Eliminate the remaining variables one-by-one via sequential color-exchange operations.

The complexity of Algorithm 1 is mainly determined by two parts: parallel processing and sequential processing. According to the analyses in Section 3.2, the parallel processing includes $O(\frac{\Delta-1+\delta}{\Delta-1+\delta(1+\log|V|)}\log|V|)$ iterations. In each iteration, each vertex executes at most Δ color exchanges. On the other hand, even if there is deadlocks during the parallel processing, the number of deadlocked variables is

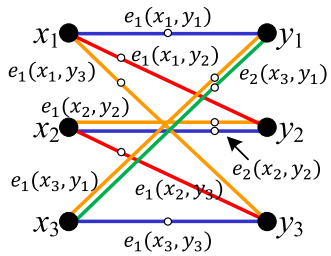


Fig. 12. Graph formulation of call requests for the fault-tolerant Clos network shown in Fig. 2a.

extremely small, as Fig. 10 displays. It follows that the complexity of the sequential processing is on the order of $O(\log |V|)$ [27]. Thus, the total time complexity is $O(\frac{\Delta(\Delta-1+\delta)}{\Delta-1+\delta(1+\log|V|)}\log|V|)$. It is easy to see that the total time complexity of the parallel algorithm is small when $\delta > 0$, and increases when δ decreases. This implies that the parallel complex coloring algorithm can take advantage of redundant colors.

4 Parallel Routing Algorithms

In this section, we apply Algorithm 1 in Section 3 to design a fast routing algorithm for fault-tolerant Clos networks. We first give a specific description of our routing algorithm for fault-tolerant Clos networks in Section 4.1, and then a detailed performance evaluation is presented in Sections 4.2 and 4.3. Specifically, we compare our algorithm with other routing algorithms in terms of time complexity in Section 4.2, and study the parallelism of our algorithm when the number of processor changes in Section 4.3.

In the simulation of performance evaluations, we assume that Clos network is fully loaded, that is, all inputs are busy. Furthermore, our simulations are implemented in the C++ programming process and all parameters are reasonably set up according to practical systems. To be more specific, we assume that these algorithms are running on the commercial central processing units (CPUs), say Intel Xeon Processor C5000/C3000 Series, which can achieve 20 floating-point operations per nanosecond (ns) and have been widely used in Cisco switches [34]. Therefore, the average running time is calculated as the ratio of the number of floating-point operations required for a route assignment to 20GFLOPS.

4.1 Parallel Routing Algorithm

We consider a symmetric fault-tolerant Clos network $C(m, n, r)$, where $m > n$. Given a set of connection requests, the corresponding bipartite graph $G = (X \cup Y, E)$ is constructed as follows. The vertex x_i in X represents the IM i, the vertex y_j in Y denotes the OM j, and the edge $e_d(x_i, y_j) \in E$ stands for the dth connection request that is originated from IM i and destined to OM j, where $i, j = 1, 2, \ldots, k$ and $d = 1, 2, \ldots, n$. An example of a corresponding bipartite graph of $C(4, 3, 3)$ in Fig. 2a is shown in Fig. 12. For instance, $e_2(x_2, y_2)$ represents the second connection request from IM 2 destined to OM 2. Let $C = \{c_1, c_2, \ldots, c_m\}$ be the set of colors, each of which corresponds to a CM in the middle stage. For example, as Fig. 12 shows, the edge $e_1(x_2, y_3)$ is colored by color c_1 (red), which indicates that a connection request from IM 2 is connected to OM 3 through the first central module, as shown in Fig. 2a.

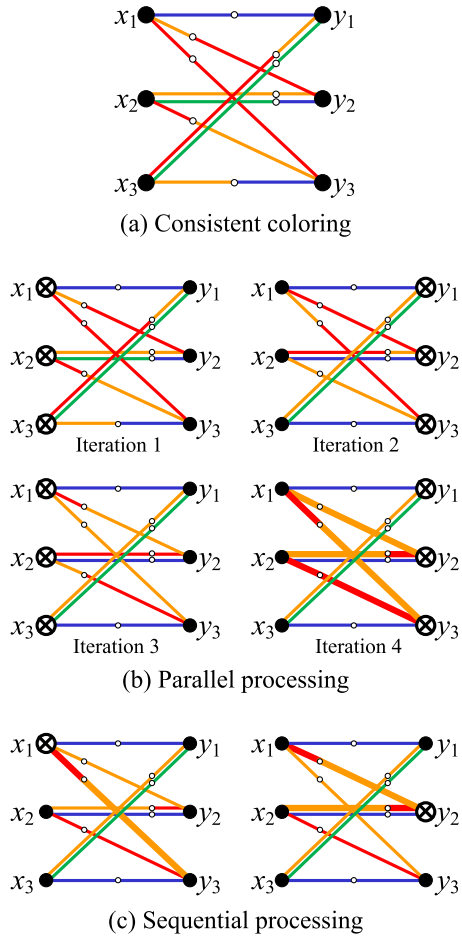


Fig. 13. Coloring process of a bipartite graph.

The route assignment in a fault-tolerant Clos network is to provide an available optical switch for connecting each connection request. To accomplish this goal, the routing algorithm starts with a bipartite graph, which is constructed from a set of connection requests inputted to the fault-tolerant Clos network. Next, the algorithm carries out Algorithm 1 such that a proper coloring can be obtained as fast as possible. After a properly colored bipartite graph is obtained, edges with the same color constitute a matching, which is mapped to a connection pattern of the corresponding CM. The routing algorithm for fault-tolerant Clos networks is described as follows.

Step 1 Initialization: Each IM and each OM implement the graph formulation, according to the current connection requests.

Step 2 Perform Algorithm 1, and return a proper coloring of the bipartite graph.

Step 3 All connections are established according to the properly colored bipartite graph G.

Since all deadlock variables can be eliminated by the sequential complex coloring, all connection requests in fault-tolerant Clos networks can be assigned a set of conflict-free paths determined by the coloring of the returned bipartite graph G.

Fig. 13 shows an example to illustrate the procedure of our routing algorithm. This example is derived from the fault-tolerant Clos network shown in Fig. 2a. A consistent

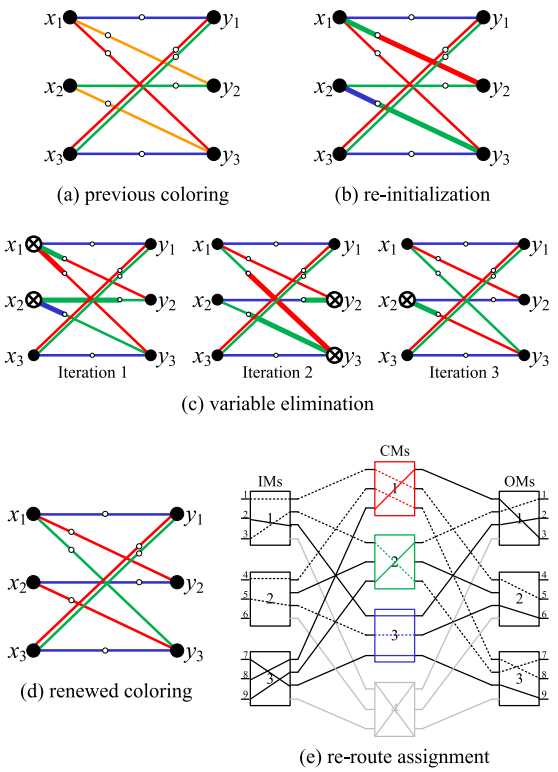


Fig. 14. Re-coloring process of a bipartite graph.

coloring of its corresponding bipartite graph is displayed in Fig. 13a. Since the number of CMs in Fig. 2a is 4, this 3-edge-colorable bipartite graph is initialized with 4 different colors. To eliminate the variables, simultaneous color-exchange operations are executed on vertexes x_1, x_2, x_3 and y_1, y_2, y_3 alternatively, as shown in Fig. 13b. In this case, two deadlock variables are left, which can be eliminated one-by-one using sequential complex coloring, as Fig. 13c shows. The resulting coloring is proper as displayed in Fig. 12. Thus, all connections are established according to this proper coloring shown in Fig. 2a.

A fault-tolerant routing algorithm should also provide a quick recovery for the switching system during switch faults. That is, when one or more optical central modules go down, the routing algorithm should be able to re-route the affected connection requests via other fault-free paths, as soon as possible. Due to the rearrangeability property of complex coloring, as mentioned in Section 2.2, our routing algorithm provides a powerful re-routing mechanism for fault-tolerant Clos networks, when some optical CMs are out of order. Instead of recoloring the whole bipartite graph, colors assigned to the removed edges are released and the new added edges are initialized with the available colors. If the new consistent coloring contains any variables, they can all be eliminated within their corresponding two-colored subgraphs by parallel processing of color-exchange operations.

An example is plotted in Fig. 14 to illustrate the re-route process. Fig. 14a shows the previous coloring of the Clos network. Assume that the fourth optical central module breaks down at this moment. Thus, the corresponding c_4 (orange) is no longer available and all edges colored by c_4 in Fig. 14a, which are $e_1(x_1, y_2)$ and $e_1(x_2, y_3)$, should be recolored by other available colors. Fig. 14b presents a new consistent coloring of these two edges. Most of the time, new variables will be introduced and can be eliminated by color exchanges shown in Fig. 14c. After that, a renewed proper coloring is obtained in Fig. 14d and thus new connections are established accordingly, as dotted lines in Fig. 14e.

4.2 Complexity Performance

The complexity of our routing algorithm is mainly dominated by the running time of Algorithm 1 whose complexity is $O(\frac{\Delta(\Delta-1+\delta)}{\Delta-1+\delta(1+\log|V|)}\log|V|)$, as we show in Section 3.3. Note that $\Delta = n, \delta = m-n$ and $|V| = 2r$ in the fault-tolerant Clos network $C(m, n, r)$ under consideration. The total running time of our routing algorithm is on the order of $O(\frac{n(m-1)}{m-1+(m-n)\log r}\log r)$. Note that if $n = r$ and $N = nr$ is the network size, the time complexity of our algorithm is $O(\frac{\sqrt{N}(m-1)}{m-1+(m-\sqrt{N})\log N}\log N)$.

Table 1 presents a complexity comparison of our work and other routing algorithms. As this table shows, our routing algorithm achieves the lowest complexity without constraints. More importantly, our algorithm is the only one that not only makes a full use of CMs but also gives an explicit relationship between the extra CMs and time complexity.

To visualize the comparison of time complexity, we compare our algorithm with two routing algorithms, Karol's algorithm [24] and the new decomposition algorithm [21], through simulation. Karol's algorithm [24] is simple and can be implemented in a parallel manner. At the worst case, it requires $O(r)$ iterations and $O(m)$ operations in each iteration. Thus, the complexity of Karol's algorithm is $O(mr)$. Meanwhile, the new decomposition algorithm in [21] can only be operated in a sequential manner and the complexity is $O(nr^2)$ at the worst case.

Fig. 15 displays the simulation results of the running time of routing algorithms with respect to the number of IMs (or OMs) r, where $n = 32$ and $m = 33$. It is clear that the required running time of our algorithm increases very slowly, compared to the other two algorithms. Specifically, the running time of our algorithm goes up from 10.28 to 19.51 ns for computing a route assignment when the IMs

TABLE 1
Comparison of Routing Algorithms for the Fault-Tolerant Clos Network $C(m, n, r)$

Research Work	Time Complexity	Parallel	Methodology
Karol's [24]	$O(mr)$	Yes	Heuristic
Lee and Liew's [26]	$O(\log m \times \log nr)$ (m is an integer power of two)	Yes	Equivalence class
Generic Rearrangement Routing [19]	$O(\|LC\|\log nr + (nr)^{1.5})$ ($\|LC\|$: #of faults)	No	Maximum matching
FT-DSRR [20]	$O(nm)$	Yes	Heuristic
Decomposition algorithm [21]	$O(nr^2)$ when $m = n$	No	Matrix decomposition
Our work	$O(\frac{n(m-1)}{m-1+(m-n)\log r}\log r)$	Yes	Complex coloring

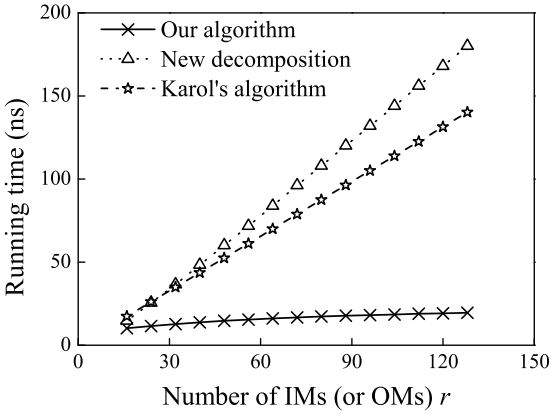


Fig. 15. Running time comparison of the routing algorithms for computing one route assignment for $n = 32$ and $m = 33$.

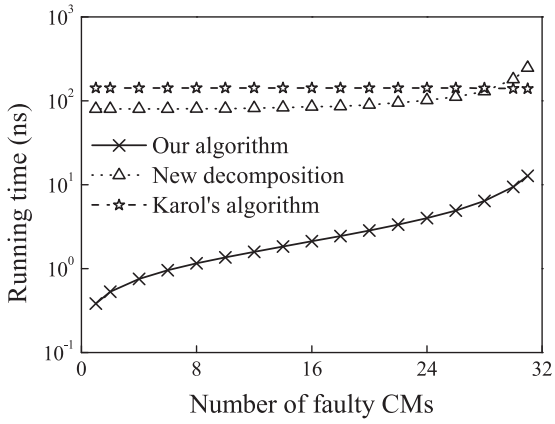


Fig. 17. Recovery time of different routing algorithms when there are CM failures in $C(63, 32, 128)$.

(or OMs) r increases from 16 to 128, while that of Karol's algorithm increases from 17.43 to 140.19 ns and that of the new decomposition algorithm from 14.73 to 180.17 ns.

Fig. 16 compares the running time of our algorithm and the other two algorithms with respect to the number of optical CMs when $r = 128$ and $n = 32$. Fig. 16 shows that the running time of our algorithm declines dramatically from 26.17 to 3.62 ns with the increase in the redundant CMs. Meanwhile, the running time of Karol's algorithm slightly increases with the number of extra CMs. This implies that more CMs even burden the routing process for Karol's algorithm. The new decomposition algorithm [21] shows the same property as ours that its running time decreases when the extra CMs are increased. However, this algorithm still costs a much longer running time than ours with all values of m, which is attributed to the fact this algorithm is a sequential algorithm.

In addition, we also make comparisons of existing algorithms in terms of the recovery time of interrupted connections when CM failures occur. The recovery time of an interrupted connection defined herein is the time from this connection is disconnected due to a CM failure to the time that the connection is re-established. In our simulation, we consider the Clos network $C(63, 32, 128)$ with 31 extra CMs as an example, in which each CM may fail with an equal probability, and the total number of faulty CMs should be less than 32 such that all possible interrupted connections can be recovered. Due to the rearrangeability of complex coloring, our algorithm uniformly outperforms the other two existing algorithms, as Fig. 17 shows.

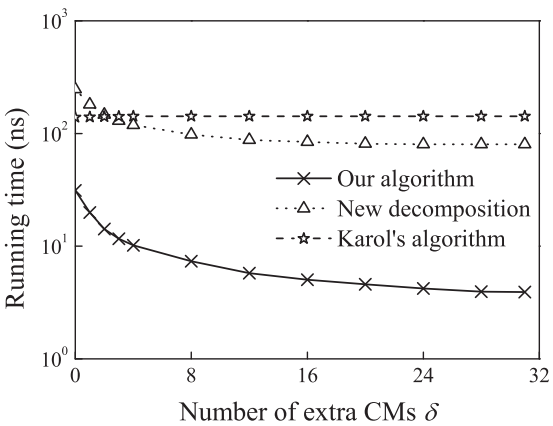


Fig. 16. Running time comparison of the routing algorithms for computing one route assignment with $r = 128$ and $n = 32$.

4.3 Scalability of Parallelism

The complexity analyses offered in Sections 3 and 4.2 assume that each IM/OM is equipped with a processor to fully parallelize our algorithm. In this section, we study the parallelizability of our algorithm by varying the number of processors, denoted by u. Specifically, we consider two scalability criterions defined as follows:

1) Strong scaling [35]: refers to the running time of a parallel algorithm versus the number of processors u for a fixed problem size. Ideally, if an algorithm assigns an evenly divided computation task with a fixed size to u parallel processors, then the strong scaling implies that the running time will be inversely proportional to the number of processors u, as the dotted link in Fig. 18a shows.
2) Weak scaling [36]: refers to the running time of a parallel algorithm versus the number of processors u for a constant amount of work per processor. Ideally, u parallel processors can solve a problem that is u times bigger in the same amount of time for any u. That is, the weak scaling implies that the running time of solving a u times bigger problem should be a constant when u grows up, as the dotted line in Fig. 19 displays.

In the following, we evaluate our algorithm in respect to these two types of scalability criterions.

4.3.1 Strong Scaling

To investigate the strong scaling aspect of our algorithm, we take the Clos network $C(33, 32, 128)$ as an example. We assume that all the inputs are busy such that the scale of the route assignment problem is fixed. We increase the number of processors u from 1 to 128 in the simulation. Clearly, when $u < 128$, multiple IMs/OMs have to share one processor and thus their color exchange operations can only be implemented in a sequential manner. When $u = 128$, our algorithm is completely parallelized and the running time is minimized.

As expected, the simulation result in Fig. 18a displays that the running time of our algorithm decreases as the

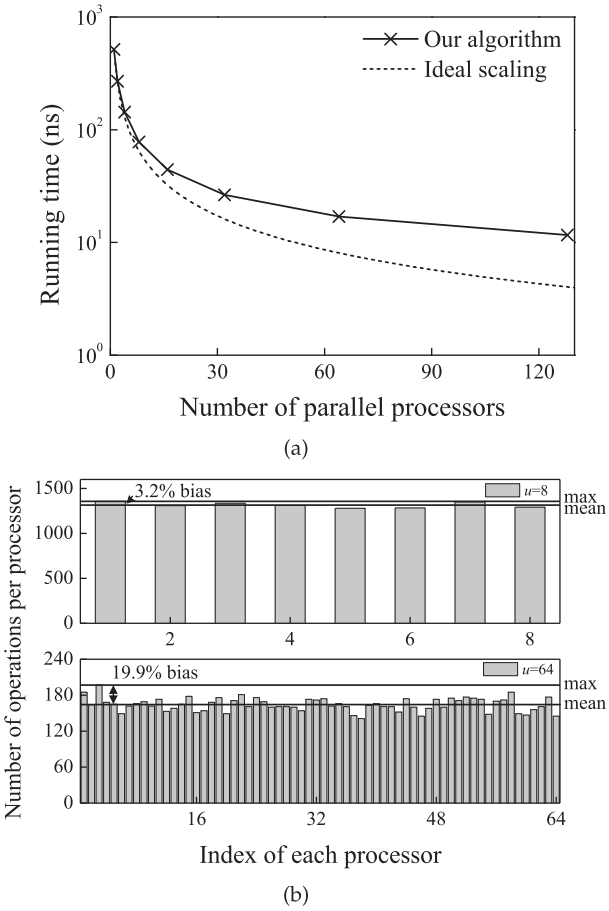


Fig. 18. Strong scaling study for our algorithm (a) Running time and (b) Distribution of the workload on each processor.

number of processors u grows. However, the decline rate is slower than $1/u$, the ideal decline rate. In particular, the gap between two curves widens with the growth of u. The running time discrepancy is due to the fact that the number of color exchanges carried out by different processors in each iteration is not evenly distributed, while the running time of each iteration is determined by the processor with the maximum workload. The number of color exchanges of each IM/OM is actually a random variable ranging from 0 to 32 in each iteration. When u is small, a number of IMs/OMs share one processor and the number of color exchanges of a processor is the sum of that of these IMs/OMs. In this case, the unevenness of workload among processors is smoothed out. On the other hand, if a dedicated processor is used by each individual IM/OM when u is large, then the unevenness of workload among processors can be significant. As Fig. 18b shows, this point is illustrated by the two workload distributions for $u = 8$ and 64, respectively.

4.3.2 Weak Scaling

As for the weak scaling aspect, we implement our algorithm on a Clos network where: a) the number of ports of each IM/OM input $n = 32$; b) the number of CMs $m = 33$; and c) each IM/OM has k processors. As the number of IMs/OMs r increases from 16 to 128, the total number of processors u will increase accordingly. When $r = 16$, each processor carries out 48 iterations. As we described in Section 3, the number of iterations is on the order of $O(\log r)$ for a fixed n and m. Thus, to keep the workload of each processor unchanged when r increases, we should install $k = O(\log r)$ processors in each IM/OM. As Fig. 19 shows, the running time of our algorithm is a logarithmic function of u, instead of a constant such as in the ideal case. Since different iterations of our algorithm are running in a sequential manner, though we installed $O(\log r)$ processors on each IM/OM, they cannot be fully used in parallel. As a result, when u increases with r, the running time increases accordingly.

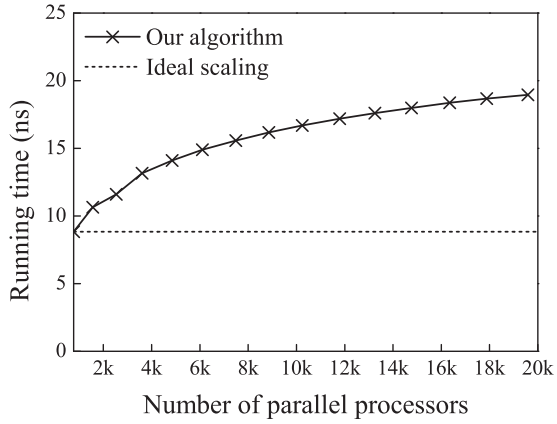


Fig. 19. Running time with different u under the weak scaling condition.

5 Conclusions

In this paper, we study the route assignment in a fault-tolerant Clos network, which is the basic component of OTN switches. This route assignment problem can be formulated as an edge-coloring problem of a bipartite graph. Based on a new algebraic edge coloring method, called complex coloring, we propose a parallel and distributed routing algorithm. Extra switch modules in the middle stage of fault-tolerance Clos networks provide redundant colors for the edge coloring. Because of those redundant colors, a new feature arises that *don't care edges* are spread on each vertex of the graph and provide a good complementation for variable elimination, which helps to produce a highly expeditious variable elimination process. The time complexity of our routing algorithm is on the order of $O(\frac{\sqrt{N}(m-1)}{m-1+(m-\sqrt{N})\log N}\log N)$ for a fault-tolerant Clos network with the size of N ports and m CMs. As the number of extra switch modules $m - \sqrt{N}$ increases, the time complexity of our algorithm can be substantially improved along with the reliability of the switching systems. Furthermore, our algorithm can provide a quick recovery from a faulty system.

Acknowledgments

This work was supported by the National Science Foundation of China under Grant 61671286, Grant 61571288, and Grant 61433009.

References

[1] T. J. Seok, N. Quack, S. Han, and M. C. Wu, "50×50 digital silicon photonic switches with MEMS-actuated adiabatic couplers," in *Proc. Opt. Fiber Commun. Conf. Exhib.*, Mar. 2015, pp. 1–3.

[2] W. M. Mellette, G. M. Schuster, G. Porter, G. Papen, and J. E. Ford, "A scalable, partially configurable optical switch for data center networks," *J. Lightw. Technol.*, vol. 35, no. 2, pp. 136–144, Jan. 2017.
[3] W. Mo, S. Zhu, Y. Li, and D. C. Kilper, "Dual-wavelength source based optical circuit switching and wavelength reconfiguration in multi-hop ROADM systems," *Opt. Exp.*, vol. 25, no. 22, pp. 27 736–27 749, Oct. 2017.
[4] P. Iovanna, F. Cavaliere, F. Testa, S. Stracca, G. Bottari, F. Ponzini, A. Bianchi, and R. Sabella, "Future proof optical network infrastructure for 5G transport," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 8, no. 12, pp. B80–B92, Dec. 2016.
[5] A. D. L. Oliva, X. C. Perez, A. Azcorra, A. D. Giglio, F. Cavaliere, D. Tiegelbekkers, J. Lessmann, T. Haustein, A. Mourad, and P. Iovanna, "Xhaul: Toward an integrated fronthaul/backhaul architecture in 5G networks," *IEEE Wireless Commun.*, vol. 22, no. 5, pp. 32–40, Oct. 2015.
[6] B. Skubic, G. Bottari, A. Rostami, F. Cavaliere, and P. Öhlén, "Rethinking optical transport to pave the way for 5G and the networked society," *J. Lightw. Technol.*, vol. 33, no. 5, pp. 1084–1091, Mar. 2015.
[7] "China Telecom's requirements on 5G transport," White Paper, China Telecom Beijing Research Institute, 2017.
[8] "OTN 5G transport network," White Paper, Huawei Technologies Co., Ltd, 2017.
[9] "C-RAN: The road towards green RAN," White Paper, China Mobile Research Institute, 2013.
[10] S. Namiki, T. Kurosu, K. Tanizawa, J. Kurumida, T. Hasama, H. Ishikawa, T. Nakatogawa, M. Nakamura, and K. Oyamada, "Ultrahigh-definition video transmission and extremely green optical networks for future," *IEEE J. Sel. Topics Quantum Electron.*, vol. 17, no. 2, pp. 446–457, Mar. 2011.
[11] S. Aleksi,"Analysis of power consumption in future high-capacity network nodes," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 1, no. 3, pp. 245–258, Aug. 2009.
[12] V. Eramo, M. Listanti, R. Sabella, and F. Testa, "Definition and performance evaluation of a low-cost/high-capacity scalable integrated OTN/WDM switch," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 4, no. 12, pp. 1033–1045, Dec. 2012.
[13] S. Okamoto, A. Watanabe, and K. I. Sato, "Optical path cross-connect node architectures for photonic transport network," *J. Lightw. Technol.*, vol. 14, no. 6, pp. 1410–1422, Jun. 1996.
[14] "Cisco network convergence system 4000 series," White Paper, Cisco Systems, Inc., 2014.
[15] B. Wu, S. Qiu, Z. Feng, S. Cao, H. Zhao, J. Xiang, C. Ding, G. N. Liu, N. Deng, and Q. Xiong, "Green and agile petabit optical sub-wavelength switching prototype for the future OTN multi-chassis switch cluster," in *Proc. Opt. Fiber Commun. Conf./Nat. Fiber Optic Eng. Conf.*, 2013, Art. no. OM3A.3.
[16] L. Wosinska, L. Thylen, and R. P. Holmstrom, "Large-capacity strictly nonblocking optical cross-connects based on microelectrooptomechanical systems (MEOMS) switch matrices: Reliability performance analysis," *J. Lightw. Technol.*, vol. 19, no. 8, pp. 1065–1075, Aug. 2001.
[17] G. I. Papadimitriou, C. Papazoglou, and A. S. Pomportsis, "Optical switching: Switch fabrics, techniques, and architectures," *J. Lightw. Technol.*, vol. 21, no. 2, pp. 384–405, Feb. 2003.
[18] K. Chen, X. Wen, X. Ma, Y. Chen, Y. Xia, C. Hu, Q. Dong, and Y. Liu, "Toward a scalable, fault-tolerant, high-performance optical data center architecture," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2281–2294, Aug. 2017.
[19] Y. Yang and J. Wang, "A fault-tolerant rearrangeable permutation network," *IEEE Trans. Comput.*, vol. 53, no. 4, pp. 414–426, Apr. 2004.
[20] K. Liu, J. Yan, and J. Lu, "Fault-tolerant cell dispatching for onboard space-memory-memory Clos-network packet switches," in *Proc. IEEE 16th Int. Conf. High Perform. Switching Routing*, Jul. 2015, pp. 1–6.
[21] H. Y. Lee, F. K. Hwang, and J. D. Carpinelli, "A new decomposition algorithm for rearrangeable Clos interconnection networks," *IEEE Trans. Commun.*, vol. 44, no. 11, pp. 1572–1578, Nov. 1996.
[22] H. Nassar and J. D. Carpinelli, "Design and performance of a fault tolerant Clos network," in *Proc. Conf. Inf. Sci. Syst.*, Mar. 1995, pp. 810–815.
[23] R. Cole, K. Ost, and S. Schirra, "Edge-coloring bipartite multigraphs in $O(E\log D)$ time," *Combinatorica*, vol. 21, no. 1, pp. 5–12, Jan. 2001.
[24] M. J. Karol and C.-L. I, "Performance analysis of a growable architecture for broad-band packet (ATM) switching," *IEEE Trans. Commun.*, vol. 40, no. 2, pp. 431–439, Feb. 1992.
[25] E. Lu and S. Q. Zheng, "Parallel routing algorithms for nonblocking electronic and photonic switching networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 8, pp. 702–713, Aug. 2005.
[26] T. T. Lee and S. Y. Liew, "Parallel routing algorithms in Benes-Clos networks," *IEEE Trans. Commun.*, vol. 50, no. 11, pp. 1841–1847, Nov. 2002.
[27] T. T. Lee, Y. Wan, and H. Guan, "Randomized Δ-edge colouring via exchanges of complex colours," *Int. J. Comput. Math.*, vol. 90, no. 2, pp. 228–245, 2013.
[28] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. E. Ng, M. Kozuch, and M. Ryan, "c-Through: Part-time optics in data centers," in *Proc. ACM SIGCOMM Conf.*, 2010, pp. 327–338.
[29] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *Proc. ACM SIGCOMM Conf.*, 2010, pp. 339–350.
[30] H. J. Chao, Z. Jing, and S. Y. Liew, "Matching algorithms for three-stage bufferless Clos network switches," *IEEE Commun. Mag.*, vol. 41, no. 10, pp. 46–54, Oct. 2003.
[31] L. Wang, T. Ye, T. T. Lee, and W. Hu, "A parallel complex coloring algorithm for scheduling of input-queued switches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 7, pp. 1456–1468, Jul. 2018.
[32] D. B. West, *Introduction to Graph Theory*, vol. 2. Upper Saddle River, NJ, USA: Prentice Hall, 2001.
[33] A. Fronczak, P. Fronczak, and J. A. Hołyst, "Average path length in random networks," *Phys. Rev. E*, vol. 70, no. 5, Nov. 2004, Art. no. 056110.
[34] "Cisco nexus 5000 series architecture: The building blocks of the unified fabric," White Paper, Cisco, 2009.
[35] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc. Spring Joint Comput. Conf.*, 1967, pp. 483–485.
[36] J. L. Gustafson, "Reevaluating Amdahl's law," *Commun. ACM*, vol. 31, no. 5, pp. 532–533, May 1988.

Lingkang Wang received the BS degree in communication engineering from the University of Electronic Science and Technology of China, Chengdu, in 2013. She is currently working toward the PhD degree in the State Key Laboratory of Advanced Optical Communication Systems and Network, Shanghai Jiao Tong University, Shanghai. Her research interests include complex coloring and its application in switching systems.

Tong Ye (M'07) received the BS and MS degrees from the University of Electronic Science and Technology of China, Chengdu, China, in 1998 and 2001, respectively, and the PhD degree in electronics engineering from Shanghai Jiao Tong University, Shanghai, China, in 2005. He was with the Chinese University of Hong Kong for one and half years as a post-doctoral research fellow. He is currently an associate professor with the State Key Laboratory of Advanced Optical Communication Systems and Networks, Shanghai Jiao Tong University. His research interests include the design of optical network architectures, optical network systems and subsystems, and silicon-ring-based optical signal processing. He is a member of the IEEE.

Tony T. Lee (SM'88-F'08) received the BSEE degree from National Cheng Kung University, Taiwan, and the MS and PhD degrees in electrical engineering from the Polytechnic Institute of NYU, Brooklyn, New York. From 2010 to 2017, he was a Zhiyuan chair professor with the Electronics Engineering Department, Shanghai Jiao Tong University. From 1993 to 2013, he was a chair professor with the Information Engineering Department, Chinese University of Hong Kong. From 1991 to 1993, he was a professor of electrical engineering with the Polytechnic Institute of NYU. He was with AT&T Bell Laboratories, Holmdel, New Jersey, from 1977 to 1983. He was with Telcordia Technologies, Morristown, New Jersey, from 1983 to 1993. He is currently a full professor with the School of Science and Engineering, Chinese University of Hong Kong (Shenzhen), and an emeritus professor of information engineering with the Chinese University of Hong Kong. He has received many awards, including the 1989 Leonard G. Abraham Prize Paper Award from the IEEE Communication Society, the 1999 Outstanding Paper Award from the IEICE of Japan, and the 1999 National Natural Science Award from China. He has served as an editor of the *IEEE Transanctions on Communications*, and an area editor of the *Journal of Communication Network*. He is a fellow of the IEEE and HKIE.