Optimal Scheduling Algorithms in WDM Optical Interconnects with Limited Range Wavelength Conversion Capability

Zhenghao Zhang, *Student Member*, *IEEE*, and Yuanyuan Yang, *Senior Member*, *IEEE*

Abstract—Optical communication is a promising candidate for many emerging networking and parallel/distributed computing applications because of its huge bandwidth. *Wavelength Division Multiplexing (WDM)* is a technique that can better utilize the optical bandwidth by dividing the bandwidth of a fiber into multiple wavelength channels. In this paper, we study optimal scheduling algorithms to resolve output contentions in bufferless time slotted WDM optical interconnects with wavelength conversion ability. We consider the general case of limited range wavelength conversion with arbitrary conversion capability, as limited range wavelength conversion is easier to implement and more cost effective than full range wavelength conversion, and it also includes full range wavelength conversion as a special case. We first consider the conversion scheme in which each wavelength can be converted to multiple wavelengths in an interval of wavelengths and the intervals for different wavelengths are "ordered." We show that the problem of maximizing network throughput can be formalized as finding a maximum matching in a bipartite graph. We then give an optimal scheduling algorithm called *the First Available Algorithm* that runs in $O(k)$ time, where k is the number of wavelengths per fiber. We also study the case where the connection requests have different priorities. We formalize the problem as finding an optimal matching in a weighted bipartite graph and give a scheduling algorithm called *the Downwards Expanding Algorithm* that runs in $O(kD + Nk\log(Nk))$ time where N is the number of input fibers of the interconnect and D is the conversion degree. Finally, we consider the circular symmetrical wavelength conversion scheme and give optimal scheduling algorithms for nonprioritized scheduling in $O(Dk)$ time and prioritized scheduling in $O(k^2 + Nk\log(Nk))$ time.

Index Terms—Wavelength-division-multiplexing (WDM), optical interconnects, scheduling, wavelength conversion, limited range wavelength conversion, bipartite graphs, bipartite matching, matroid.

1 Introduction and Background

Many emerging networking applications, such as databrowsing in the World Wide Web, video conferencing, video on demand, E-commerce, and image distribution, require very high network bandwidth, often far beyond what today's high-speed networks can offer. Optical networking is a promising solution to this problem because of the huge bandwidth of optics: A single fiber has a bandwidth of nearly 50 THz [14]. To fully utilize the bandwidth, a fiber is divided into a number of independent channels, with each channel on a different wavelength. This is referred to as *wavelength-division-multiplexing (WDM)* [1].

In a WDM all-optical network, data is modulated on a selected wavelength channel and this information-bearing signal remains in the optical domain throughout the path from source to destination. A wavelength converter can be used to convert one wavelength to another wavelength, and make the network more flexible for satisfying various connection requests. Studies show that network performance is greatly improved by using wavelength converters [3]. The converters can be *full range* which are capable of converting a wavelength to any other wavelengths, or *limited range* which only convert a wavelength to several adjacent wavelengths and the number of these adjacent wavelengths is called the *conversion degree*. Full range wavelength converters are quite difficult and expensive to implement due to technological limitations [12], [10]. Limited range wavelength converters, on the other hand, are much cheaper and easier to implement and can achieve network performance similar to full range wavelength converters even when the conversion degree is very small [12], [10], [13]. Thus, limited range converters are considered as a practical, cost-effective choice for providing wavelength conversion ability in WDM networks, which will be the main focus of this paper. Note that full range wavelength conversion can be viewed as a special case of limited range wavelength conversion when the conversion degree is equal to the number of wavelengths on a fiber.

We study scheduling algorithms in *WDM optical interconnects* (also called WDM switch or crossconnect in the literature) with limited range wavelength conversion in this paper. A WDM optical interconnect provides interconnections between a group of input fiber links and a group of output fiber links with each fiber link carrying multiple wavelength channels. Such an interconnect can be used to provide high-speed interconnections among a group of processors in a parallel and distributed computing system or serve as an optical crossconnect (OXC) in a wide-area communication network. We consider WDM optical interconnects that operate synchronously, such as time slotted WDM packet switching networks where information is carried by optical packets that arrive at the interconnect at the beginning of time slots [11]. In such an interconnect, scheduling algorithms are needed to smartly allocate the resources (the wavelength channels) to the requests (the arrived packets) to optimize network performance, such as

• *The authors are with the Department of Electrical and Computer Engineering, State University of New York at Stony Brook, Stony Brook, NY 11794-2350. E-mail: {zhhzhang, yang}@ece.sunysb.edu.*

Manuscript received 2 July 2003; revised 2 Mar. 2004; accepted 7 May 2004. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0104-0703.

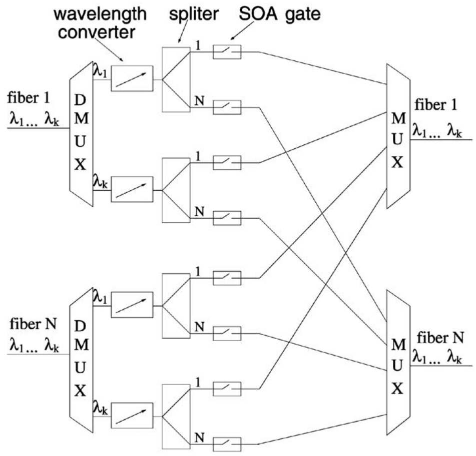


Fig. 1. A wavelength convertible WDM optical interconnect.

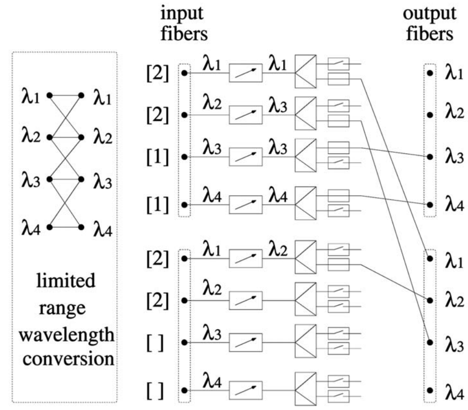


Fig. 2. Requests and wavelength channel assignments of an example interconnect with two input/output fibers and four wavelength per fiber. The number in the parenthesis are the destination of a request.

network throughput. Since optical buffers are currently made of fiber delay lines and are very expensive and bulky [2], we consider bufferless WDM optical interconnects in this paper. We refer to an incoming packet as a connection request or simply a request. We consider unicast traffic, i.e., each connection request has only one destination fiber. The duration of a request can be one time slot or several time slots, however, as in [11], [23], we focus on the one time slot case since in most packet switching networks the interconnect only switches fixed length cells.

Such an interconnect is shown in Fig. 1. It has input N fibers and N output fibers. On each fiber, there are k wavelengths that carry independent data. Thus, there are a total of Nk input wavelength channels and Nk output wavelength channels. It can be seen from the figure that an input fiber is first fed into a demultiplexer, where different wavelength channels are separated from one another. An input wavelength is then fed into a wavelength converter to be converted to a proper wavelength. The output of a wavelength converter is then split into N signals, which are connected to each of the output fibers under the control of N SOA gates. The signal can reach the output fiber if the SOA gate is on, otherwise, it is blocked. Since the request has only one destination, only one of the SOA gates is on at a time. In the front of each output fiber there is an optical combiner which multiplexes the signals on different wavelengths into one composite signal and send to the output fiber. Apparently, it is required that all signals to the optical combiner must be on different wavelengths.

To understand the problem that needs to be solved by the scheduling algorithm, we can use the following example. Consider a simple interconnect with two input/output fibers and four wavelengths per fiber shown in Fig. 2. Suppose under limited range conversion, wavelength $\lambda_i, 1 \leq i \leq 4$, can be converted to λ_j, where $j \in [\max(i-1,1), \min(i+1,4)]$, as shown in the left part of the figure. At the beginning of a time slot, there are four connection requests on λ_1, λ_2, λ_3, and λ_4 arrived at input fiber 1, destined for output fiber 2, 2, 1, and 1, respectively. In the figure, the destination of a request is the number shown in the parenthesis. There are two connection requests on λ_1 and λ_2 arrived at input fiber 2, and all destined for output fiber 2. We first notice that there is no contention at output fiber 1, since there are only two requests destined to it, and they are on different wavelengths. These two requests can both be granted and no wavelength conversion is needed. However, there is contention at output fiber 2 since there are four requests, two on λ_1 and two on λ_2, destined for this output fiber. Without wavelength conversion, one request on each of the wavelengths must be dropped. With limited range wavelength conversion, three wavelengths, λ_1 to λ_3, can be converted from λ_1 and λ_2 and, therefore, three of the four requests destined for output fiber 2 can be granted. We can assign channel λ_1 to the request arrived at input fiber 1 on λ_1, assign channel λ_2 to the request arrived at input fiber 2 on λ_1, assign channel λ_3 to the request arrived at input fiber 1 on λ_2, and reject the request arrived at input fiber 2 on λ_2. Based on these decisions, the wavelength converters are configured to convert input wavelengths to proper output wavelengths, as shown in the figure. An SOA gate is set to on if the request is granted and is destined to the output fiber connected to the gate. We can see in the example that, when contention arises at an output fiber, to maximize network throughput, we attempt to find the largest group of requests that are contention free.

Extensive research has been conducted on scheduling algorithms for various electronic switches (which can be considered as a single wavelength switch). For example, [5] and [6] considered scheduling algorithms in input-buffered electronic switches under unicast traffic. Scheduling algorithms for WDM broadcast and select networks were also well studied in recent years, see, for example, [15], [16]. In this type of network, the source node broadcasts its information to all other nodes via a selected wavelength, and only the destination node tunes into this wavelength to get the message. In this way, only one wavelength on the fiber is used at a time, both for the source and the destination node. It is a different type of network from the WDM interconnect considered in this paper. We consider a space-division switch where all wavelengths on a fiber can be utilized simultaneously. There has also been some work in the literature on the performance analysis of WDM optical interconnects with limited range wavelength conversion in WDM wavelength routing networks (or optical circuit switched networks), e.g., [10], [12], [13]. Note that connection requests arrive at this type of optical interconnect asynchronously and, thus, there is no need for a scheduling algorithm since the requests have a natural order and are assumed to be served in a "first come first served" manner. Time slotted WDM switches with limited range wavelength conversion were considered in [23], in which a simple scheduling algorithm was given. However, the authors did not show whether the algorithm is optimal in terms of maximizing network throughput. In this paper, we will present scheduling algorithms that can maximize network throughput. We will also consider networks that supports Quality of Service (QoS) and will give optimal algorithms that both maximize network throughput and give service differentiation.

The rest of this paper is organized as follows: Section 2 describes two types of limited range wavelength conversions, namely, the "ordered interval" wavelength conversion and the "circular symmetrical" wavelength conversion. Section 3 introduces the request graph and shows that the problem of maximizing network throughput is equivalent to the problem of finding a maximum matching in the request graph. Section 4 gives the First Available Algorithm for finding maximum matchings in the request graph for ordered interval wavelength conversion. Section 5 gives the Downwards Expanding Algorithm for finding optimal matchings in the request graph when the connection requests have different priorities. In Section 6, we consider circular symmetrical wavelength conversion and give the algorithms for finding maximum matchings and optimal matchings in the request graphs for both nonprioritized and prioritized scheduling. Section 7 gives some simulation results of the algorithms and, finally, Section 8 concludes the paper.

2 Wavelength Conversion

2.1 Ordered Interval Wavelength Conversion

All-optical wavelength conversion is usually achieved by conveying information from the input light signal to a probe signal [8], [9]. The probe signal is generated by a tunable laser tuned to the desired output wavelength. The tuning range of the laser is continuous, but under limited range wavelength conversion, it is only part of the whole spectrum because of constraints such as tuning speed, loss, etc. We can see that a wavelength can be converted to an interval of wavelengths, because the tuning range of the laser covers an interval of wavelengths. Also, note that, if the laser for the conversion of λ_1 can be tuned to λ_3, then the laser for the conversion of λ_2 should also be able to be tuned to λ_3 since λ_2 is closer to λ_3 than λ_1 is. Thus, we have following two observations on wavelength conversion:

Observation 1. *The wavelengths that can be converted to by λ_i for $i \in [1, k]$ can be represented by interval $[begin(i), end(i)]$, where $begin(i)$ and $end(i)$ are positive integers in $[1, k]$. We call wavelengths that belong to this interval* adjacency set *of λ_i.*

Observation 2. *For two wavelengths λ_i and λ_j, if $i < j$, $begin(i) \leq begin(j)$ and $end(i) \leq end(j)$.*

We call this type of wavelength conversion "ordered interval" because the adjacency set of an wavelength can be represented by an interval of integers, and the intervals for different wavelengths are "ordered." The cardinality of the adjacency set is called the *conversion degree* of the wavelength. Different wavelengths may have different conversion degrees. The conversion degree of the interconnect, denoted by D, is defined as the largest conversion degree of all wavelengths. The *conversion distance* of a wavelength is defined as the largest difference between a wavelength and a wavelength that can be converted from it.

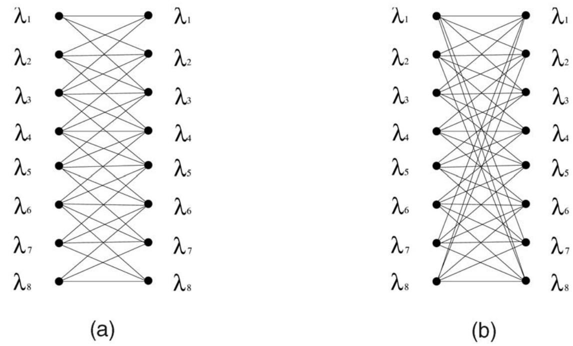


Fig. 3. Conversion graphs for an 8-wavelength optical system for two types of wavelength conversions, both with conversion distance 2. (a) Ordered interval. (b) Circular symmetrical.

We can use a bipartite graph to visualize the wavelength conversion, as have informally practiced in the example in Fig. 2. Let the left side vertices represent input wavelengths and the right side vertices represent output wavelengths. λ_i on the left and λ_j on the right are connected if λ_i can be converted to λ_j. Fig. 3a shows such a conversion graph for $k = 8$. The adjacency set of λ_3, for example, can be represented as $[1, 5]$, and the conversion degree of λ_3 is 5. Other wavelengths, for example, λ_1, has a smaller conversion degree of 3. Since 5 is the largest conversion degree, $D = 5$. The conversion distance for λ_3 is $3 - 1 = 2$. In fact, the conversion distance is 2 for all the wavelengths in this example.

Note that the observations we made about wavelength conversion above are very general, only relying on the two facts observed at the beginning of this section. It is allowed that different wavelengths have different conversion degrees and different conversion distances. This type of wavelength conversion is also used in other research work, for example, [24], [25]. In fact, the conversion distance in [24], [25] is the same for all wavelengths and therefore is a special case of the wavelength conversion considered in this paper. Full range wavelength conversion can also be considered as a special case, by letting the conversion degrees for all wavelengths be k.

2.2 Circular Symmetrical Wavelength Conversion

The ordered interval wavelength conversion discussed above is not the only type of wavelength conversion used in the literature. Another popular type of wavelength conversion which is vastly used for the purpose of performance analysis can be called "circular symmetrical" wavelength conversion, in which the conversion degrees of all wavelengths are the same [12], [10], [13]. A wavelength can be converted to d lower wavelengths and d higher wavelengths, where d is the conversion distance. For the wavelengths near the "boundary," say, λ_1, it is allowed to be converted to wavelengths on the other end, say, λ_k. To be specific, wavelength λ_i can be converted to $[k+i-d, k] \cup [1, i+d]$ for $1 \leq i \leq d$, $[i-d, i+d]$ for $d+1 \leq i \leq k-d$, and $[i-d, k] \cup [1, d+i-k]$ for $k - d < i \leq k$. Fig. 3b shows a conversion graph for circular symmetrical wavelength conversion when $k = 8$ and $D = 5$. From an implementational point of view, circular symmetrical wavelength conversion is not as practical as the ordered interval wavelength conversion, but in some cases, it may simplify the theoretical analysis.

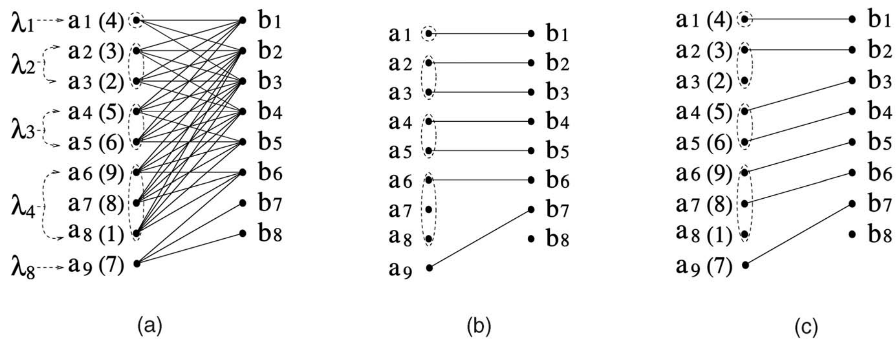


Fig. 4. Request graphs and matchings when the request vector is $[1, 2, 2, 3, 0, 0, 0, 1]$ in an 8-wavelength interconnect with conversion distance 2. (a) Request graph. The numbers in the parenthesis are the weights of the requests. (b) Maximum matching. (c) Optimal matching.

3 Problem Formalization

In this section, we show how the problem of maximizing network throughput in the WDM interconnect can be formalized into a bipartite graph matching problem. We first consider the case where all connections hold for one time slot. The multitime slot case will be discussed at the end of Section 4.

First, we consider one output fiber. The relationship between the connection requests destined for an output fiber and the available wavelength channels on this output fiber can be represented by a bipartite graph, called *request graph*. The left side vertices represent the connection requests and the right side vertices represent output wavelengths. The vertices on each side of the graph are placed according to their wavelength indices, λ_1 first, then λ_2, then λ_3, and so on. Vertices on the same wavelength can be placed in an arbitrary order. There is exactly one vertex on each wavelength on the right side. However, on the left side, there could be more than one vertices on the same wavelength since there may be more than one requests on the same wavelength going to the same output fiber. We will use $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_m$ to denote the left side vertices and right side vertices, respectively, throughout the paper. There is an edge connecting left side vertex a_i and right side vertex b_u if the wavelength of a_i can be converted to the wavelength represented by b_u which is λ_u. The conversion graph discussed earlier can be simply considered as a special case of the request graph when there is exactly one connection request coming on each wavelength.

For convenience, we also define the *request vector*. A request vector is a $1 \times k$ row vector, with the ith element representing the number of connection requests arriving on wavelength λ_i. Fig. 4a shows a request graph for an output fiber under the ordered interval conversions when the request vector is $[1, 2, 2, 3, 0, 0, 0, 1]$. The numbers in the parenthesis are the weights of the requests which will be discussed later in Section 5.

We can also draw a "whole" request graph with the left side vertices being the all the requests arrived at the interconnect and the right side vertices being all the wavelength channels on the N output fibers. However, since a wavelength channel on output fiber p will not be assigned to a request destined to output fiber q if $p \neq q$, there will be no edge connecting this pair of vertices even if the wavelength channel is within the conversion range of the request. As a result, the whole request graph will have N isolated components, or N separate "small" request graphs, one for each output fiber. We can work on the small request graphs one by one. In other words, we can use the same scheduling algorithm to find an optimal solution for each request graph, and the N optimal solutions for the N request graphs, when put together, will be the optimal solution for the whole request graph. Therefore, in the following, we explain our algorithm for only one output fiber. The input to the scheduling algorithm is the connection requests destined to this fiber. The output of the algorithm is the decisions about whether a request is granted or not and, if granted, which wavelength channel it is assigned to. As the relations between the requests and the wavelength channels can be fully described by the request graph, the scheduling algorithm will be explained as an algorithm for a bipartite graph. The algorithm can be run independently and in a distributed manner to speed up the scheduling process.

In a request graph G, let E denote the set of edges. Any wavelength assignment can be represented by a subset of E, E', where edge $a_i b_u \in E'$ if wavelength channel b_u is assigned to connection request a_i. Under unicast traffic, any connection request needs only one output channel. Also, an output channel can be assigned to only one connection request. It follows that the edges in E' are vertex disjoint, because if two edges share a vertex, either one connection request is assigned two wavelength channels or one wavelength channel is assigned to two connection requests. Thus, E' is a *matching* in G. For a given set of connection requests, to maximize network throughput, we should find a maximum matching in the request graph. The maximum matching for Fig. 4a is shown in Fig. 4b.

The best known algorithm for finding a maximum matching in an arbitrary bipartite graph was given in [19], and has time complexity $O[n^{\frac{1}{2}}(m+n)]$, where n and m are the number of vertices and edges in the bipartite graph, respectively. If we directly adopt this algorithm in our scheduling algorithm, the time complexity would be as high as $O(N^{\frac{3}{2}}k^{\frac{3}{2}}D)$ since the number of left side vertices in a request graph alone could be as large as Nk and each left side vertex is adjacent to up to D right side vertices. However, faster algorithms are required for scheduling in WDM optical interconnects as the decision must be made in real time within a time slot, which is in the order of μs [11]. We will show that the request graph for limited range wavelength conversion exhibits some nice properties so that faster algorithms are possible.

TABLE 1
List of Symbols in Section 4

G	:	request graph.
E	:	edge set of the request graph.
n	:	number of left side vertices.
m	:	number of right side vertices.
$[begin(a_i), end(a_i)]$	:	adjacency interval of left side vertex a_i.
$[begin'(b_u), end'(b_u)]$	:	adjacency interval of right side vertex b_u.
$MATCH[]$	:	output of the First Available Algorithm.
$current$	:	location of the right side vertex immediately follows the right side vertex just matched by the First Available Algorithm.

4 Maximum Matchings in Ordered Interval Wavelength Conversion Request Graphs

In this section, we discuss how to find a maximum matching in an ordered interval wavelength conversion request graph. Table 1 lists the symbols that we are going to use in this section.

We have the following theorem concerning the properties of a request graph.

Theorem 1. *A request graph with ordered interval wavelength conversion has the following properties:*

Property 1. *The adjacency set of any left side vertex is an interval. Namely, if left side vertex a_i is adjacent to right side vertices b_u and b_v, where $u < v$, a_i is adjacent to all b_w, where $u \le w \le v$. In the following, we use interval $[begin(a_i), end(a_i)]$ to represent the adjacency set of an left side vertex a_i. We call $begin(a_i)$ and $end(a_i)$ the begin value and end value of a_i, respectively. The vertex with index $begin(a_i)$ and $end(a_i)$ are called the begin vertex and end vertex of a_i, respectively.*

Property 2. *Let $[begin(a_i), end(a_i)]$ be the adjacency set of left side vertex a_i and $[begin(a_j), end(a_j)]$ be the adjacency set of left side vertex a_j. If $i < j$, then $begin(a_i) \le begin(a_j)$ and $end(a_i) \le end(a_j)$.*

Property 3. *In the request graph G, if edge $a_ib_u \in E$, $a_jb_v \in E$ and $i < j$, $u > v$, then $a_ib_v \in E$, $a_jb_u \in E$.*

Property 4. *Properties 1 and 2 also hold for right side vertices. Namely, the adjacency set of any right side vertex b_u is also an interval or can be represented by $[begin'(b_u), end'(b_u)]$, and for two right side vertices b_u and b_v, if $u < v$, then $begin'(b_u) \le begin'(b_v)$ and $end'(b_u) \le end'(b_v)$.*

Property 5. *Removing any vertex from the request graph, all the above properties still hold.*

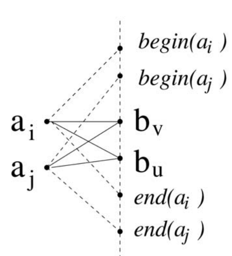
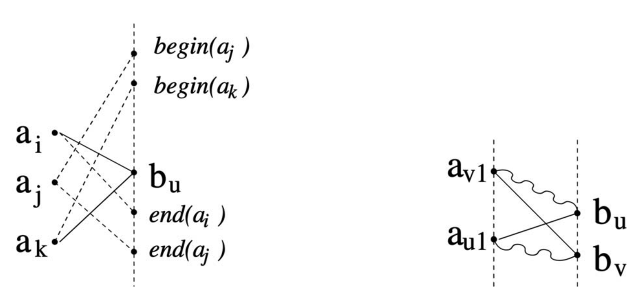


Fig. 5. Illustration of Property 3 of a request graph. If $a_ib_u \in E$ and $a_jb_v \in E$, then $a_ib_v \in E$ and $a_jb_u \in E$.

Proof. The first two properties come directly from the two observations on the ordered interval wavelength conversion. Next, we show that Property 3 holds. A visual illustration of this property is shown in Fig. 5. Let the adjacency set of a_i be $[begin(a_i), end(a_i)]$ and the adjacency set of a_j be $[begin(a_j), end(a_j)]$. By Property 1, in order to prove $a_ib_v \in E$, we need to show that $begin(a_i) \le v \le end(a_i)$. First, since $i < j$, by Property 2, we have $[begin(a_i) \le begin(a_j)]$. We also have $begin(a_j) \le v$ since $a_jb_v \in E$. Therefore, $begin(a_i) \le v$. Next, since $a_ib_u \in E$, we have $u \le end(a_i)$. Then, $v \le u \le end(a_i)$ and, thus, $a_ib_v \in E$. Similarly, we can show $a_jb_u \in E$.

We now give the proof for Property 4. We first show that the adjacency set of any right side vertex must also be an interval. Let b_u be any right side vertex. Suppose $a_ib_u \in E$ and $a_kb_u \in E$ where $i < k$, as shown in Fig. 6a. To prove this claim, we need to show that $a_jb_u \in E$ for all a_j where $i < j < k$. That is to say, $begin(a_j) \le u \le end(a_j)$ where $[begin(a_j), end(a_j)]$ is the adjacency set of a_j. We have $begin(a_k) \le u$ since $a_kb_u \in E$. Since $j < k$, by Property 2, $begin(a_j) \le begin(a_k) \le u$. Similarly, we have $u \le end(a_i)$ since $a_ib_u \in E$. Finally, since $i < j$, by Property 2, $u \le end(a_i) \le end(a_j)$. Thus, $a_jb_u \in E$.

Next, we prove that Property 2 also holds for right side vertices. We show this by contradiction. Assume Property 2 does not hold. Then, we can find two vertices b_u and b_v where $u < v$, with the adjacency set of b_u being $[begin'(b_u), end'(b_u)] = [u_1, u_2]$ and the adjacency set of b_v being $[begin'(b_v), end'(b_v)] = [v_1, v_2]$, but either $u_1 > v_1$ or $u_2 > v_2$. We show that u_1 cannot be greater than v_1, and the proof for the other case is similar. If $u_1 > v_1$, as shown



(a) (b)

Fig. 6. Property 4 of a request graph. (a) If $a_ib_u \in E$ and $a_kb_u \in E$, $a_jb_u \in E$ for $i < j < k$. (b) a_{u1}, which is the begin vertex of b_u cannot have a larger index than a_{v1} which is the begin vertex of b_v if $u < v$. The wavy line segments are the nonexisting edges.

TABLE 2
First Available Algorithm for Finding a Maximal Matching

```
First Available Algorithm
 current:=1;
 for i := 1 to n do
     if current < begin(a_i)
         MATCH[i] := begin(a_i);
         current := MATCH[i] + 1;
     else if current <= end(a_i)
         MATCH[i] := current;
         current := MATCH[i] + 1;
     else
         MATCH[i] := Λ;
     end if
 end for
```

in Fig. 6b, we have $a_{u_1}b_u \in E$ and $a_{v_1}b_v \in E$, but $u_1 > v_1$ and $u < v$. By Property 3, we have $a_{v_1}b_u \in E$, which contradicts the fact that a_{u_1} is the first left side vertex adjacent to b_u. In Fig. 6b, this nonexisting edge is drawn in a wavy line segment.

Finally, Property 5 is quite straightforward and we only need to show that, if some vertex is removed, Properties 1 and 2 still hold, since other properties can be derived from these two properties. If the vertex to be removed is a left side vertex, Properties 1 and 2 obviously hold. Hence, we only need to consider the case that the removed vertex is from the right side. Note that, if the right side vertex b_u is removed, we will need to renumber the right side vertices: The indices of vertices from b_1 to b_{u-1} are unchanged, and the indices of vertices from b_{u+1} to b_m are all decremented by 1. After this renumbering, for any interval in the old numbering, if it does not contain u, it is still an interval in the new numbering; if it contains u, it is also an interval since the indices following u are all decremented by 1. Thus, Property 1 holds. Similarly, Property 2 can also be easily verified by considering all four combinations of the adjacency sets of two left side vertices, i.e., containing u or not in both sets. □

A bipartite graph with Property 1 is called a *convex bipartite graph* and was first studied in [20]. Clearly, the request graph we consider is a convex bipartite graph. Furthermore, due to other properties of the request graph, mainly Property 2, it is a special case of a convex bipartite graph, and we call it an *ordered convex bipartite graph*. For simplicity, throughout this section and the next section, we will still refer to it as a request graph and, by a request graph, we mean a bipartite graph with Properties 1-5. There are also *doubly convex bipartite graphs* in the literature [21] in which the adjacency sets of the left side and the right side vertices are all intervals. Our request graph is also doubly convex, but is also a special case of it because there exist graphs that are doubly convex but do not satisfy all Properties 1-5.

Finding a maximum matching in a convex bipartite graph is much easier than that in general bipartite graphs. Lipski Jr. and Preparata [21] gave such an algorithm in $O(n + mA(m))$ time, where n is the number of left side vertices, m is the number of right side vertices, and $A(m)$ is a slowly growing function with respect to m. Since the request graph has stronger properties, we can obtain even simpler algorithms. We present the First Available Algorithm as described in Table 2 for finding a maximum cardinality matching in a request graph. The input to this algorithm is the adjacency set of left side vertices denoted by interval $[begin(a), end(a)]$ for each left side vertex a. The output of the algorithm is array $MATCH[]$. $MATCH[i] = j$ means that the ith left side vertex is matched to the jth right side vertex. $MATCH[i] = \Lambda$ if the ith left side vertex is not matched to any right side vertex.

In the description of the algorithm, n is the number of left side vertices. $current$ is the location of the right side vertex that immediately follows the most recently matched right side vertex. In other words, if we just matched right side vertex b_v to some left side vertex, $current = v + 1$. Initially, $current = 1$. As shall be seen soon, in this algorithm, we match left side vertex a_i to the first adjacent right side vertex (the one with the smallest index) that has not been matched to other left side vertex yet, such a vertex is called the *first available* vertex.

To show the correctness of the algorithm, we need to first prove the following invariant:

Lemma 1. *Throughout the execution, in every step, the First Available Algorithm finds the first available vertex for a left side vertex if such a vertex exists.*

Proof. By induction. This is obviously true in the first step since the first left side vertex, if not isolated, will be matched to its begin vertex which is the first available vertex by definition. If it is isolated, it has no available vertex. Now, suppose it is true for all the steps before the ith step when checking a_i, where $i > 1$. We now prove that it is still true for the ith iteration, i.e., the algorithm finds a first available vertex for a_i.

We first claim that all right side vertices with indices smaller than $current$ are not available to a_i. They are either not adjacent to a_i or are adjacent to a_i, but were already matched to some other left side vertices. Suppose this is not the case. Then, there exists a vertex b_u which is available to a_i but $u < current$. We know that b_v is matched where $v = current - 1$. Suppose b_v is matched to a_j. We have $i > j$ since the left side vertices are checked in an increasing order according to their indices. We also know that $u < v$. Since $a_jb_v \in E$ and $a_ib_u \in E$, by Property 3, we have $a_jb_u \in E$, which contradicts with the inductive hypothesis that b_v is the first available vertex for a_j.

We next claim that none of the right side vertices with indices no less than $current$ are matched yet. Since, by the algorithm, the most recently matched right side vertex has the largest index among all the matched right side vertices, and the value of $current$ is larger than the index of this most recently matched right side vertex.

Therefore, at the ith iteration, if $current < begin(a_i)$, $begin(a_i)$ is not matched and it will be the first available vertex for a_i. Otherwise, if $begin(a_i) \leq current \leq end(a_i)$, the first available vertex is simply $current$, and if $current > end(a_i)$, all the vertices in $[begin(a_i), end(a_i)]$ have been matched and there is no available vertex for a_i. □

Having seen that we always match a left side vertex to its first available vertex, we next show that by doing so we can obtain a maximum matching of the request graph.

Theorem 2. *The First Available Algorithm finds a maximum matching in a request graph.*

Proof. We first define the "top edge" of a request graph as the edge connecting the first nonisolated left side vertex to the first nonisolated right side vertex. For example, edge a_1b_1 in Fig. 4a is the top edge. We claim that the top edge must belong to some maximum matching. Let a_ib_u be the top edge of request graph G. Given any maximum matching of G, if it contains edge a_ib_u, we have found such a maximum matching. Otherwise, we show it can be transformed into a maximum matching that contains edge a_ib_u. Note that in this maximum matching at least one of a_i and b_u is matched since, otherwise, we can add edge a_ib_u in and obtain a matching with a larger cardinality. If exactly one of a_i and b_u is matched, we can add edge a_ib_u in and remove the edge covering a_i or b_u. The resulting matching is still maximum. For example, if in Fig. 4a a matching matches a_1 to b_2 and b_1 is unmatched, we can match a_1 to b_1 and leave b_2 unmatched. This new matching will be of the same cardinality as the old matching. Hence, we only need to consider the case that a_i and b_u are both matched, but not to each other. Let a_i be matched to b_v and b_u matched to a_j. We have $i < j$ and $u < v$ since a_i and b_u are the first nonisolated vertices. Thus, by Property 3, we have $a_ib_u \in E$ and $a_jb_v \in E$. Therefore, we can match a_i to b_u and a_j to b_v, and the resulting matching is still maximum and also contains edge a_ib_u. For example, if in Fig. 4a, in a matching a_1 is matched to b_2 and a_2 is matched to b_1, we can match a_1 to b_1 and match a_2 to b_2. After this change, the new matching still has the same number of edges but now contains edge a_1b_1.

Now, consider the subgraph of G obtained by deleting vertices a_i and b_u and all the edges incident to them. We call it *residual graph* of G and denote it as G_1. We assert that edge a_ib_u together with a maximum matching in G_1 is a maximum matching in G. We give a proof by contradiction. Suppose this is not true, then let M be a maximum matching in G that contains edge a_ib_u. From previous discussions, we know that such a matching always exists. Let M_1 be a maximum matching in G_1. Since $M_1 \cup \{a_ib_u\}$ is not a maximum matching in G, $\mid M \mid > 1 + \mid M_1 \mid$. However, $M_1' = M \setminus \{a_ib_u\}$ is a matching in G_1. Therefore, $\mid M_1' \mid = \mid M \mid -1 > \mid M_1 \mid$, which contradicts with the fact that M_1 is a maximum matching in G_1.

Therefore, to find a maximum matching of G, we can first take the top edge and then find a maximum matching of G_1. Note that by Property 5, finding a maximum matching in G_1 can be done in exactly the same way as in G. Thus, we can take the top edge of G_1 and go on to work on G_2 which is the residual graph of G_1. The process can be repeated and, in each step, we simply take the top edge until no vertex is left in the residual graph. By then, the edges we have taken will constitute a maximum matching of G. Note that, by matching a left side vertex to the first available right side vertex, we are precisely taking a top edge of some residual graph. This completes our proof. □

For example, Fig. 7a is G_1, the residual graph of the request graph shown in Fig. 4a after removing a_1 and b_1. In G_1, the top edge is a_2b_2. Thus, edge a_2b_2 should be added to the matching. Note that this is exactly what is done by the First Available Algorithm since, after matching a_1 to b_1, the first available vertex of a_2 will be b_2. The residual graph of G_1 after removing a_2 and b_2 is shown in Fig. 7b, and the top edge a_3b_3 should be added to the matching. This process is carried on. After adding six edges, residual graph G_6 is shown in Fig. 7c. Note that, although a_7 and a_8 have smaller indices than a_9, they are isolated and, therefore, the top edge of G_6 is a_9b_7. The First Available Algorithm will find that there is no available vertex for a_7 and a_8 and leave them unmatched and match a_9 to b_7. The maximum matching is shown in Fig. 4b.

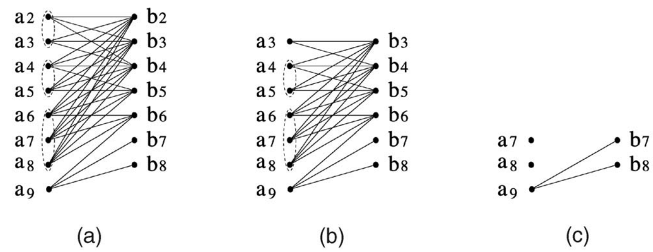


Fig. 7. Residual graphs of request graph shown in Fig. 4a. (a) G_1. (b) G_2. (c) G_6.

We now analyze the complexity of this algorithm. The loop in the algorithm is executed exactly n times and the work within the loop can be done in constant time. Thus, the running time is $O(n)$, where n is the number of left side vertices. However, due to Property 4, the same algorithm can also be run on the right side vertices. That is to say, we can also scan through the right side vertices and match them to their first available left side vertices. Therefore, if we know before hand which side has fewer vertices, we can choose to run the algorithm on that side, and the running time would be $O(\min\{n, m\})$, where m is the number of right side vertices.

For our applications, we can choose to run the algorithm on the right side since the maximum number of left side vertices can be as large as Nk when all the connection requests are destined to this fiber, while the number of right side vertices is k, the number of wavelengths on a fiber. The running time thus becomes $O(k)$. However, the scheduling time is not completely independent of network size N since to generate the input to the algorithm one might have to scan all the input channels.

Before concluding this section, we would like to address the issue when the connection requests are more then one time slot long. So far, we have only considered one time slot connection requests. Note that in most packet switched interconnects, connection requests can be considered as one time slot since the switching core only switches fixed length cells [6]. However, it poses an interesting problem when the duration of a request is multitime slot long. In this case, at the beginning of a time slot, some of the output wavelength channels may still be occupied by connections arrived earlier and cannot be assigned to the newly arrived requests. We can still draw request graphs by simply removing the right side vertices representing these occupied channels. By Property 5, we can use the same algorithm to find maximum matchings for the request graph with this incomplete right side. In this way, we maximize the number of granted requests at this time slot. This will be the optimal solution if we measure the network performance by the number of granted requests at current time slot. It might not be optimal under other criteria such as maximizing network utilization. However, since the decisions are made in real time and we do not know what

TABLE 3
List of Symbols in Section 5

M_i	:	current matching when checking a_i.
Π	:	the set of the vertices selected by the matroid algorithm.
$n(i)$	:	number of vertices in Π when checking a_i.
R	:	the set of right side vertices reachable from a_i by M_i alternating path.
R_I	:	reachable set at the I_{th} expansion.
a_{u_I}, a_{l_I}	:	left side vertices matched to R_I with smallest and largest index.
b_w	:	the unmatched right side vertex found by the Downwards Expanding Algorithm.
a_δ	:	the matched left side vertex closest to a_i satisfying $i < \delta$.
b_u	:	the vertex matched to a_δ.
M_i'	:	the matching updated from M_i and covering a_i and b_w.

requests will arrive in the future, global optimization goals such as maximizing network utilization cannot be achieved. We are interested in finding algorithms to give suboptimal solutions to this case in our future work.

5 Optimal Matchings in Ordered Interval Wavelength Conversion Weighted Request Graphs

In this section, we consider the scheduling in a communication environment that requires Quality of Service (QoS) in which the connection requests have different priorities. In this case, the interconnect should be able to give service differentiation: Lower priority connection requests should have higher blocking probabilities than higher priority connection requests. We can assign weights to connection requests based on their priorities and then find a group of contention-free connection requests with maximum total weight. We can solve this problem by generalizing the results in the previous section to the case that left side vertices of the request graph are weighted.

More formally, the problem can be described as follows: Given a request graph (which is a bipartite graph with the five properties described in Section 4) with weighted left side vertices, find a matching that maximizes both the number and the total weight of the covered left side vertices. Such a matching is called an *optimal matching*. As an example, Fig. 4c is the optimal matching for the request graph in Fig. 4a where the weights of the left side vertices were shown in the parenthesis next to them.

Table 3 lists the symbols which we are going to use in this section that are not listed in Table 1.

Next, we will adopt a useful tool, matroid, to solve this problem.

5.1 Matroid Greedy Algorithm

A matroid is a structure defined on a finite whole set S and a family of subsets of the whole set, with property usually refereed to as independence defined on the elements of the subsets [18]. For example, in a graph, we can let the vertex set be the whole set. A proper subset, which is a subset belonging to the matroid, is a group of vertices that can be covered by a matching. These vertices are said to be independent of each other in the matroid theory. Greedy algorithms can be used to find optimal solutions for problems defined on a matroid.

An optimal matching of an arbitrary bipartite graph can be found by the matroid greedy algorithm [18], [21]. In essence, the algorithm tries to find a set of vertices that can be covered by a matching by checking the vertices one by one according to their weights, vertices with larger weights first. A vertex is added to the set if it can be covered along with the previously selected vertices. To elaborate, the algorithm starts with an empty set Π. In step s, let a_i be the left side vertex with the $s\text{th}$ largest weight. The algorithm checks whether there is a matching covering a_i and all the previously selected vertices in Π. If yes, add a_i to Π, otherwise, leave a_i uncovered. Update $s \leftarrow s+1$ and repeat until all vertices have been checked. When finished, Π stores left side vertices that can be covered by an optimal matching.

For example, consider the request graph shown in Fig. 4a. In the first step, the matroid algorithm should check the left side vertex with the largest weight, which is a_6. a_6 can be matched to b_2 and, therefore, is added to Π. The next vertex needs to be checked is a_7, which can be matched to b_3, and is also added to Π. In the following steps a_9, a_5, a_4, a_1, a_2, a_3, a_8 are checked, in this order. a_3 and a_8 cannot be matched to any vertices if all the vertices added to Π prior to them should remain matched. So, when the algorithm terminates, $\Pi = \{a_6, a_7, a_9, a_5, a_4, a_1, a_2\}$. The optimal matching is shown in Fig. 4c.

The matching found by this greedy algorithm is optimal in a strong sense:

1. It is a maximum cardinality matching.
2. The total weight of the vertices covered by the matching is maximum.
3. The matching is also lexicographically maximum: Let the matching found by the greedy algorithm be M and let $a_1, a_2, \dots, a_{|M|}$ be the left side vertices covered by M sorted in a nonincreasing order according to their weights. Let M' be any other matching and let $a_1', a_2', \dots, a_{|M'|}'$ be the left side vertices covered by M' sorted in a nonincreasing order according to their weights. Then, $w(a_i') \leq w(a_i)$ for all $1 \leq i \leq| M' |$ where $w()$ is the weight of a vertex.

The key operation of the matroid algorithm is to check whether there exists a matching covering the new vertex and all the previously selected vertices. Suppose we are checking vertex a_i and let the matching covering all the vertices in Π at this step be M_i. An M_i *alternating path* is a path with edges alternating between edges belonging to M_i and edges not belonging to M_i. There is such a matching if and only if there exists an M_i alternating path with one end being a_i and the other end being an unmatched right side vertex. For example, in Fig. 8a an alternating path starts from a_2 and ends at b_7 is shown. The edges belong to

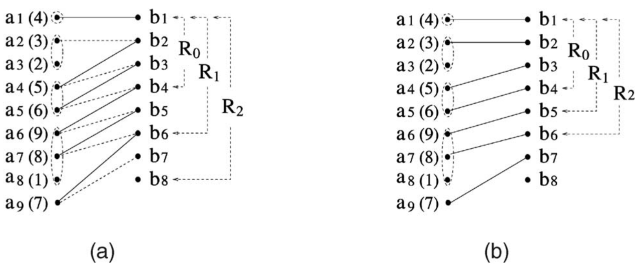


Fig. 8. (a) Expanding reachable set for checking a_2. a_2 can be matched. Solid lines are edges in matching M_2. An M_2 alternating path is also shown. (b) Expanding reachable set for checking a_3. a_3 cannot be matched.

current matching M_2 are shown in solid lines and the edges not belonging to M_2 are shown in dashed lines. If such a path is found, we can perform a "flip" operation along the path to augment the matching and also to cover a_i: Remove the edges that used to be in M_i and add in the edges that used to not be in M_i. For example, in Fig. 8a after the flip operation, the new matching is shown in Fig. 4c, which covers a_2 and has one more edge than M_2. For more detailed coverage of alternating paths, the readers are refereed to [22] and [18].

It has been shown in [21] how to find an optimal matching in a convex bipartite graph by using the matroid greedy algorithm and we will briefly describe it here. Recall that a bipartite graph is convex if it has Property 1. Define the set of the right side vertices that can be reached by a_i using M_i alternating paths as the *reachable set* of a_i and denote it by R. For example, the reachable set of a_2 in Fig. 8a is all the right side vertices. If there is an unmatched vertex in the reachable set, a_i can be matched, since there will be an M_i alternating path starting from a_i and ending at this unmatched right side vertex. R can be found by expanding itself in a step by step manner. In the first step, vertices in the adjacency set of a_i are added to R. In each step, the newly added vertices are checked to see whether there is one unmatched. If yes, we are done. Otherwise, R needs to be expanded. When expanding R, for each left side vertex which is matched to one of the vertices in R, say, a_j, add to R the right side vertices adjacent to a_j, but not in R yet, since these vertices can also be reached by a_i using M_i alternating paths (a_j can be reached by a_i using M_i alternating paths, and all the vertices adjacent to a_j can be reached from a_j). This is simply to take the unions of R and the adjacency set of a_j. Note that the adjacency set of a_j is an interval, and it has at least one common element with R, which is the vertex matched to a_j. Thus, if R is also an interval, the union of the two will also be an interval. Because in the beginning R is an interval, R will always be an interval during the expansion process. Hence, the expansion is simply to take the unions of two intervals which can be done in constant time. To find the entire reachable set, no more than $n(i)$ expansions are needed, where $n(i)$ is the number of the left side vertices in Π when checking a_i.

In a general bipartite graph, to find the reachable set or equivalently an M_i augmenting path, we may also need only $n(i)$ expansions, but the work in each expansion may not be constant time. One might need to scan all the edges incident to a left side vertex which might take as many as m operations, where m is the number right side vertices. Therefore, we can see that finding the reachable set in a convex bipartite graph is considerably easier. We next show that in a request graph, the amount of work can be further reduced.

5.2 The Downward Expanding Algorithm

We now present a new algorithm, called the Downwards Expanding Algorithm, for finding an optimal matching in the request graph.

First, we notice that, due to Property 2 of the request graph, when expanding the reachable set R, we do not have to take the union of R with the adjacency set of *all* the left side vertices matched to vertices in R, instead, only two are needed. To find R, at first we can set $R_0 = [begin(a_i), end(a_i)]$. If one of the vertices in R_0 is not covered by M_i, we can match a_i to this vertex and we are done. Otherwise, R_0 needs to be expanded. As explained earlier, to expand R_0 is to take the union of R_0 with the adjacency set of a left side vertex matched to a vertex in R_0. Of all the left side vertices matched to vertices in R_0, let a_{u_0} and a_{l_0} be the one with the smallest and the largest index, respectively. We claim that right side vertices in interval $R_1 = [begin(a_{u_0}), end(a_{l_0})]$ all can be reached by a_i using M_i alternating paths. This is because by Property 2 of the request graph, the union of the adjacency set of a_{u_0} and R_0 is $[begin(a_{u_0}), end(a_i)]$ and the union of the adjacency set of a_{l_0} and R_0 is $[begin(a_i), end(a_{l_0})]$, and these two intervals must have some overlap. Furthermore, also by Property 2, the adjacency set of any other left side vertex matched to a vertex in R_0 is a subset of R_1. Thus, there is no need to take the union of R_0 with the adjacency set of vertices other than a_{u_0} and a_{l_0}.

We can check all the right side vertices in R_1 that have not been checked before. If there is an unmatched vertex, we are done. Otherwise, we can again find two left side vertices with the smallest index and the largest index that are matched to vertices in R_1, say, a_{u_1} and a_{l_1}, and expand R_1 to $R_2 = [begin(a_{u_1}), end(a_{l_1})]$. This process is repeated until an unmatched right side vertex is found, or until the interval cannot be expanded further which means after some Ith expansion $R_{I-1} = R_I$. In this case, we have found all the right side vertices reachable from a_i via M_i alternating paths.

Before moving on, we first define *crossing edges* in a request graph. In a request graph G, we say that edge a_ib_u and a_jb_v *cross* each other if $i < j$ and $u > v$. Note that by Property 3 of request graphs, if edge a_ib_u and a_jb_v cross each other, then $a_ib_v \in E$ and $a_jb_u \in E$ and these two edges are not crossing each other. For example, in Fig. 4a, edge a_1b_2 and a_2b_1 are a pair of crossing edges. $a_1b_1 \in E$ and $a_2b_2 \in E$ and are not crossing each other. A matching is called "noncrossing" if it does not have any crossing edges. In such a matching, the jth matched left side vertex is matched to the jth matched right side vertex. There always exists a maximum matching in a request graph that is noncrossing since given any maximum matching, if there are crossing edges, we can simply use Property 3 to replace the two crossing edges with two noncrossing edges until no crossing edges are left.

Now, we show that, if the current matching M_i has some properties, finding an unmatched right side vertex can be greatly simplified. To be specific, the desired properties are: 1) the matching is noncrossing and 2) no matched left side vertices has an unmatched upper neighbor, where an unmatched upper neighbor of a matched left side vertex

a_j is defined as an unmatched adjacent right side vertex with a smaller index than the vertex matched to a_j. For example, in Fig. 4a, at the first step of the algorithm when checking a_6, if it is matched to b_4, it will have two unmatched upper neighbors, b_2 and b_3. If it is matched to b_2, it will have no unmatched upper neighbor.

The first simplification is about searching for a_{u_I} and a_{l_I}. We will not need to compare the indices of all the left side vertices matched to vertices in R_I, instead, we can simply find b_{u_I} and b_{l_I} which are the vertices in R_I with the smallest index and the largest index, respectively, and a_{u_I} must be the vertex matched to b_{u_I} and a_{l_I} must be the vertex matched to b_{l_I}, because the matching is noncrossing. Note that finding b_{u_I} and b_{l_I} is very easy as they are simply the beginning and the end of interval R_I.

The second simplification, as implied by the name of the Downwards Expanding Algorithm, is that we need only to expand the reachable set downwards. In other words, we only take the union of R_I with the adjacency set of a_{l_I}, and a_{u_I} is not needed. This will need a little proof. Suppose the claim is not true, that is, an unmatched right side vertex can be found by searching right side vertices with smaller indices than $begin(a_i)$. Suppose an unmatched vertex b_w is found after the $(I+1)\mathrm{th}$ expansion by an upward search, that is to say, b_w is in the adjacency set of a_{u_I}, where a_{u_I} is the left side vertex matched to R_I with the smallest index, and $b_w \notin R_I$. b_w must have a smaller index than b_v, the vertex matched to a_{u_I} since $b_v \in R_I$. But, this contradicts the fact that a_{u_I} does not have unmatched upper neighbors.

The algorithm is shown in Table 4. The *while* loop corresponds to the searching for an unmatched vertex. We can see that when checking a_i, we start with $begin(a_i)$ and search downward in interval $[x_0, y_0] = [begin(a_i), end(a_i)]$. If no unmatched right side vertex is found, we find the left side vertex matched to $end(a_i)$ and let it be a_{l_0}. Then, start searching from x_1 in interval $[x_1, y_1] = [y_0 + 1, end(a_{l_0})]$. Again, if no unmatched right side vertex is found, start the search from x_2 in interval $[x_2, y_2] = [y_1 + 1, end(a_{l_1})]$, where a_{l_1} is the vertex matched to b_{y_1}. This process is repeated until an unmatched vertex is found or at some step I $x_I > y_I$, in the latter case $a_{l_{I-1}}$ is matched to its end vertex and the reachable set cannot be expanded. Note that, in the algorithm, the expanding is to set R_{I+1} to $R_I \cup$ $[begin(a_{l_I}), end(a_{l_I})]$, i.e., expanding only downward. And, to find a_{l_I} we used the fact that the current matching is noncrossing.

As an example, consider the request graph in Fig. 4a when checking a_2. The expanding process is shown in Fig. 8a. The current matching M_2 is shown in solid lines. We can see that it is noncrossing, and no matched left side vertices have an unmatched upper neighbor. At the beginning, the algorithm finds that the adjacency set of a_2 is $R_0 = [1, 4]$. But, currently, b_1 to b_4 are all matched, therefore, R_0 needs to be expanded. a_{l_0} is a_6 which is the vertex matched to b_4. The adjacency set of a_6 is $[2, 6]$. Therefore, the algorithm sets $x_1 = 4 + 1$, $y_1 = 6$, and starts searching from b_5 in interval $[5, 6]$. It finds that b_5 and b_6 are also matched. It then finds a_{l_1}, which is the vertex matched to b_6 and is a_9. The adjacency set of a_9 is $[6, 8]$. Thus, it starts searching from b_7 in interval $[x_2, y_2] = [6 + 1, 8]$. It finds that b_7 is unmatched. Hence, it returns $b_w = b_7$. Fig. 8b shows the expanding for a_3. In this case, no unmatched vertices can be found.

TABLE 4
Downward Expanding Algorithm for Finding an Optimal Matching

Downwards Expanding Algorithm
$\Pi := \emptyset$;
for $s := 1$ **to** n **do**
  Let a_i be the left side vertex with the s_{th} largest weight
  $[x, y] := [begin(a_i), end(a_i)]$
  while $x \leq y$
    for $w := x$ **to** y **do**
      if b_w is not matched
        exit the while loop;
      end if;
    end for;
    let a_l be the vertex matched to b_y
    $[x, y] := [y + 1, end(a_l)]$
  end while;
  if b_w is found
    Find a_δ in Π such that a_δ is closest to a_i and has a larger index than a_i
    if no such a_δ
      $MATCH[i] := w$
    else
      $u := MATCH[\delta]$
      if $w < u$
        $MATCH[i] := w$
      else
        for $t := u$ **to** $w - 1$ **do**
          let a_h be the vertex matched to b_u
          $MATCH[h] := MATCH[h] + 1.$
        end for
        $MATCH[i] := u$
      end if
    end if
    $\Pi := \Pi \cup \{i\}$
  end if
end for

We notice the following facts: 1) $w \geq begin(a_i)$ since the algorithm only scans vertices with indices no less than $begin(a_i)$, and 2) right side vertices with indices in range $[begin(a_i), b_{w-1}]$ are all matched, in the case that b_w is not the begin vertex of a_i.

If an unmatched right side vertex b_w is found, we need to update matching M_i to M_i' to cover a_i and b_w. In the algorithm, after b_w was found, we search for the matched left side vertex with a larger index than a_i and closest to a_i. If no such vertex exists, we simply match a_i to b_w. Otherwise, let this vertex be a_δ, and suppose a_δ is matched b_u. If $w < u$, we also match a_i to b_w. If $w > u$, we match a_i to b_u and perform the following shifting operation: For the left side vertices matched to b_u to b_{w-1} under M_i, increment the indices of their matchings by 1. In this case, we say a_i is matched by shifting. In the other case, when a_i is matched to b_w and the matchings for all other vertices are not changed, we say that a_i is directly matched.

As an example, consider Fig. 8a. When checking a_2, the algorithm finds an unmatched right side vertex b_7. In current matching M_2, a_4 is the matched left side vertex closest to a_2 and has a larger index. a_4 is matched to b_2. Therefore, $w = 7$, $\delta = 4$, and $u = 2$. Since $2 < 7$, a_2 should be matched to b_2. For all the vertices matched to b_2 to b_6 under

M_2, which are a_4, a_5, a_6, a_7, a_9, the indices of the vertices matched to them are incremented by 1. Matching M_2' is shown in Fig. 4c which is the optimal matching of the request graph since vertices that are to be checked in the following steps cannot be covered.

To show that the algorithm is correct, we need first to show that the new matching given by the algorithm is valid, i.e., no vertex is matched to a vertex not adjacent to it. To see this, first consider when a_i is directly matched. In this case, the matchings for all other vertices are not changed, and we need only to show that a_i is adjacent to b_w. We prove this by contradiction. Suppose a_i is not adjacent to b_w, then either $w < begin(a_i)$ or $w > end(a_i)$. But, since $w \geq begin(a_i)$, we have $w > end(a_i)$. Since right side vertices with indices in $[begin(a_i), w-1]$ are all matched, right side vertices in $[begin(a_i), end(a_i)]$ must be also matched. Some of them must be matched to left side vertices with a larger indices than a_i since, if this is not true, i.e., all the vertices in $[begin(a_i), end(a_i)]$ are matched to vertices with smaller indices than a_i, the downward expansion must have stopped at the first iteration and would not have found b_w since a_{l_0}, which is the vertex matched to the end vertex of a_i, must also have end value of $end(a_i)$. Then, we have found a contradiction since if a_i is directly matched, either there is no matched left side vertex with a larger index than a_i or the one closest to a_i is matched to a vertex with a larger index than b_w and, therefore, also larger than $end(a_i)$.

If a_i is matched by shifting, we need to show that every left side vertex involved in the shifting is adjacent to the right side vertex assigned to it. First, we show that b_u, the vertex used to matched to a_δ, is adjacent to a_i. Since by Property 2 of the request graph, b_u must have a larger index than $begin(a_i)$ since it is matched to a_δ and $\delta > i$. Therefore, if b_u is not adjacent to a_i, $u > end(a_i)$. Since the matching is noncrossing, all the left side vertices with larger indices than i must be matched to right side vertices with larger indices than $end(a_i)$. This is a contradiction since some of the vertices in $[begin(a_i), end(a_i)]$ must be matched to left side vertices with larger indices than i, as shown a short while ago. Next, we show that all other involved left side vertices cannot be matched to its end vertex under M_i and, therefore, each of them is adjacent to the right side vertex immediately following the one matched to it. To see this, suppose b_w was found in the $(I+1)$th expansion. Then, b_w must be adjacent to a_{l_I}. By Property 2 of the request graph, all the left side vertices involved in the shifting with larger indices than a_{l_I} must also adjacent to b_w and, therefore, is not matched to their end vertices. For the left side vertices involved in the shifting with smaller indices than a_{l_I}, if some of them, say, a_h is matched to its end vertex, the algorithm will have stopped expanding at a_h and would have not expanded to a_{l_I}.

Now, we show that M_i' is also noncrossing, again by contradiction. First, consider when a_i is directly matched to b_w. In this case, the matchings for all other left side vertices are not changed. Therefore, if M_i' has crossing edges, it must be that the newly added edge a_ib_w is crossing some other edges. But, a_ib_w cannot cross any edges covering left side vertices with larger indices than a_i since, if a_i is matched to b_w, either there is no matched left side vertices with larger indices than a_i, or they are all matched to right side vertices with larger indices than b_w. Thus, if a_ib_w crosses another edge a_jb_v, it must be $i > j$ and $w < v$. By Property 3 of request graph, a_j is adjacent to b_w. This contradicts the fact that under M_i, a_j has no unmatched upper neighbor.

The proof for the case when a_i is matched by shifting is similar. First, notice that after the shifting the new edges are not crossing each other. Suppose in M_i', b_w is matched to a_l. The new edges cannot cross edges covering vertices with larger indices than a_l since they are matched to vertices with larger indices than b_w. Therefore, if they are crossing, it must be the new edges are crossing edges covering vertices with smaller indices than a_i. Following exactly the same argument as in the previous case, we can find a contradiction.

Finally, we show that under M_i', no matched left side vertex has an unmatched upper neighbor. If a_i is directly matched, one more right side vertex becomes matched, and the matchings for all other vertices are not changed. Therefore, for all the matched left side vertices except a_i, if no one has an unmatched upper neighbor under M_i, it must also be the case under M_i'. Therefore, we need only to show that under M_i' a_i has no unmatched upper neighbor. Note that this is obviously true if b_w is the begin vertex of a_i. The claim is also true if b_w is not the begin vertex of a_i, since vertices in $[begin(a_i), w-1]$ are all matched under M_i'.

If a_i is matched by shifting, suppose b_w is matched to a_l in M_i'. Again, it is simple to verify that none of the matched left side vertices with smaller indices than a_i or with larger indices than a_l has an unmatched upper neighbor. Also, by Property 2 of the request graph, if a_i has no unmatched upper neighbor, none of the matched vertices from a_i to a_l can have an unmatched upper neighbor. Thus, what is left to show is that a_i has no unmatched upper neighbor. This can be shown in exactly the same way as in the case when a_i is directly matched.

Therefore, we have the following theorem.

Theorem 3. *The Downward Expanding Algorithm finds an optimal matching in the request graph.*

5.3 Complexity Analysis

Now, we analyze the complexity of this algorithm. The algorithm is composed of two parts: 1) expanding the reachable set and 2) updating the matching. In 1), there are two cases: Either a) the algorithm cannot find an unmatched right side vertex, or b) it finds such a vertex. We show that the time spent in a) can be controlled under $O(k)$. To see this, suppose when checking a_i, the algorithm finds that the reachable set cannot be expanded any further after the Ith expansion. In this case, the algorithm finds that a_{l_I} is matched to its end vertex. We notice the fact that for any other vertex which is to be checked later, if it is within the wavelength range starting from the wavelength of a_i and and ending at the wavelength of a_{l_I}, it also cannot find an unmatched right side vertex. Therefore, we can mark the wavelengths in this range. If a left side vertex is in this range or the expansion reaches this range, there is no need to expand the reachable set any further. In other words, the expansion needs to be carried on only if it does not hit any such marked wavelengths. As a result, for any expanding that ends without finding an unmatched vertex, it only visits those wavelengths that have not been marked before. And, after the failure to find an unmatched vertex, the wavelengths that are involved in the expansion that were previously unmarked should also be marked. Since there are a total of k wavelengths, the time of a) is bounded by $O(k)$.

In the algorithm, the time spent in b) is no more than that in 2). We next show that the time spent in 2) is bounded by $O(kD)$, where D is the conversion degree. When updating the matching, the first thing is to find a_δ, the matched vertex closest to a_i and is with a larger index than a_i. To find a_δ, we can start from the wavelength of a_i and check whether there is an matched vertex on higher wavelengths. We need to check only D wavelengths since if the wavelength of a_δ is greater than the wavelength of a_i by an amount of D, b_u cannot be adjacent to a_i. In our application, since there k right side vertices, 2) is performed no more than k times. Therefore, the time spent in finding a_δ is bounded by $O(kD)$.

When updating the matching, if a_i is matched to b_w, the matchings for all other vertices are not changed, and it only takes constant time. We next show that time spent in shifting is bounded by $O(kD)$. This is because when doing the shifting, the matchings for all the left side vertices must shifted down by one and will never go up. Therefore, for any left side vertex, it can be involved in this shifting for no more than D times. There can be at most k matched left side vertex, and the time is thus $O(kD)$.

As a conclusion of the complexity analysis, the Downward Expanding Algorithm runs in $O(kD)$ time, where k is the number of wavelength on a fiber and D is the conversion degree. However, note that in order to find the optimal scheduling, we should first sort the left side vertices according to their weights since the algorithm must check the vertices with larger weights first. The sorting will need $O(Nk\log(Nk))$ time in the worst case when all the left side vertices have different weights. Thus, the total scheduling time is $O(kD + Nk\log(Nk))$.

6 Scheduling Algorithms for Circular Symmetrical Wavelength Conversion

In this section, we consider the scheduling when the wavelength conversion is circular symmetrical. We can still draw the request graphs and formalize the problem as a matching problem. However, the request graphs in this case no longer exhibit the nice properties as the ordered interval wavelength conversion. For example, the adjacency set of some left side vertex can no longer be represented by an interval. Fortunately, we can still consider it as an extended case to the ordered interval wavelength conversion. For example, the adjacency set of a left side vertex a_i can be represented either as an interval $[begin(a_i), end(a_i)]$ or the union of two intervals: $[begin(a_i), m-1] \cup [0, end(a_i)]$, where m is the number of right side vertices. Note that in this type of request graph, we number the vertices as $a_0, a_1, \ldots, a_{n-1}$ and $b_0, b_1, \ldots, b_{m-1}$ because of the circular symmetrical nature. The ideas for the ordered interval wavelength conversion can still be used here. We will describe how to find maximum matchings and optimal matchings in this type of request graph. The proofs for these algorithms are similar to the ordered interval case and will not be repeated here, rather, we will only outline the general ideas.

6.1 Finding Maximum Matching for Nonprioritized Scheduling

We first consider the nonprioritized scheduling. In the request graph, we refer to an edge connecting a left side vertex to a right side vertex at the other end as a "wrap around" edge. If there is no wrap around edge, the request graph would be the same as the ordered interval case. In a request graph, if the first left side vertex a_0 has degree D and has t wrap around edges connecting to $b_{m-t}, b_{m-t+1}, \ldots, b_{m-1}$, we can redraw the request graph by rotating these right side vertices up: Let b_{m-t} be b_0, b_{m-t+1} be $b_1, \ldots, b_{m-1}$ be b_t. The old b_0 would be b_{t+1}. It can be considered as circularly shifting these vertices to the other end while still keeping all the edges. The adjacency set of a_0 would be $[0, D-1]$. Call this request graph G_0. Note that any request graph can be transformed to this type of request graph, and there will be no wrap around edges incident to the first left side vertex. Hence, from now on we will only consider the request graph of this type.

Clearly, there must be a maximum matching M of G_0 in which a_0 is matched. Suppose a_0 is matched to b_u. If in M there is some left side vertex that is matched to a right side vertex with a larger index than b_u using a wrap around edge, say, a_i is matched to b_v where $v > u$ and a_ib_v is a wrap around edge, similar to the ordered interval case, we can show that edges a_0b_v and a_ib_u exist, and we can match a_0 to b_v and match a_i to b_u. Also, if in M there is a left side vertex that is matched to a right side vertex with a smaller index than b_u, say, a_j is matched to b_w where $w < u$, but a_jb_w is not a wrap around edge, we can similarly match a_0 to b_w and match a_j to b_u. Eventually, we obtain a maximum matching in which a_0 is matched to some b_u, and all the matched right side vertices with smaller indices than b_u are matched to some left side vertices via wrap around edges.

If we know before hand such a maximum matching M, we can circularly shift vertices $b_0, b_1, \ldots, b_{u-1}$ down to the other end. Call this new request graph G_u. In G_u, matching M has no wrap around edges. Let G'_u be the subgraph of G_u which is obtained by deleting all the wrap around edges. Then, the maximum matching of G'_u is the maximum matching of G_u. Notice that the maximum matching of G'_u can be found by the First Available Algorithm.

Of course, we do not know before hand this maximum matching of G_0. But, the reasoning above can still lead us to a new algorithm. We can generate D isomorphic graphs, $G_0, G_1, \ldots, G_{D-1}$ of the original request graph G_0 by circularly shifting down 0 vertex, 1 vertex, 2 vertices, up to $D-1$ vertices starting from b_0 to b_{D-2}. Then, we can delete all the wrap around edges of these graphs and generate D subgraphs, $G'_0, G'_1, \ldots, G'_{D-1}$. Now, we can use the First Available Algorithm for finding maximum matchings for $G'_0, G'_1, \ldots, G'_{D-1}$ and the one with the maximum cardinality is the maximum matching of G_0. We call the operation of deleting the wrap around edges as "breaking" the request graph, and the algorithm is thus called the Breaking Algorithm.

Since, in our applications, the degree of a left side vertex cannot exceed D which is the conversion degree and the running time of the First Available Algorithm is $O(k)$, the Breaking Algorithm runs in $O(kD)$ time.

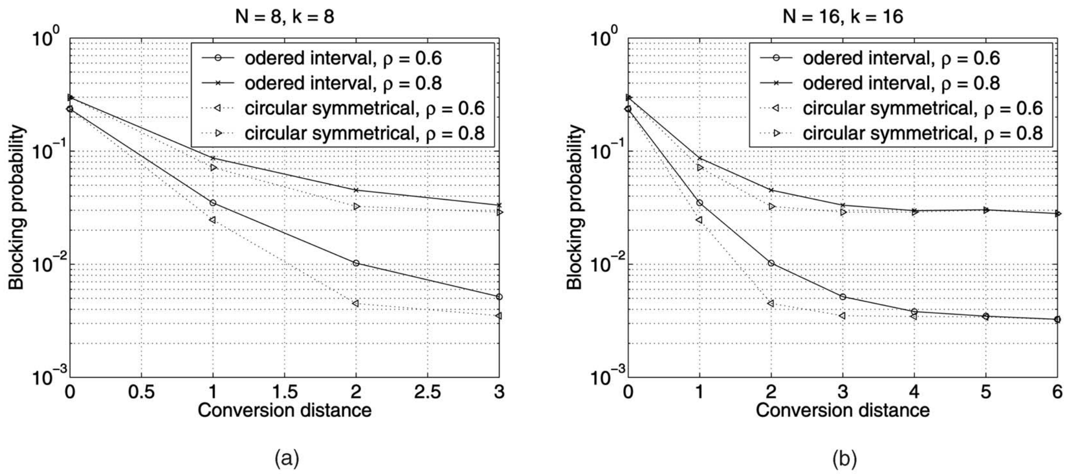


Fig. 9. Blocking probability of WDM interconnects under bursty traffic where the connection requests have no priority. (a) 8×8 interconnect with eight wavelengths per fiber. (b) 16×16 interconnect with 16 wavelengths per fiber.

6.2 Finding Optimal Matching for Prioritized Scheduling

For the circular symmetrical wavelength conversion, optimal matchings for the prioritized scheduling can also be found relatively easily. The idea is still to use the interval property to find the reachable set for a left side vertex. The adjacency set of a left side vertex is, of course, not always an interval, and sometimes may be the union of two intervals. We refer to the former as type 1 adjacency set and the latter as type 2 adjacency set. The union of two adjacency sets that share at least one element is still either type 1 or type 2.

Now, suppose we are using the matroid algorithm in Section 5.1 to find the reachable set for a_i. At the beginning, the reachable set R_0 is simply the adjacency set of a_i. Thus, it is either type 1 or type 2 interval. When expanding R_0 to R_1, we basically take unions of two adjacency sets that share one element. Thus, R_1 is also either type 1 or type 2 interval. This is true for all the following expansions and the reachable sets we find are all type 1 or type 2 intervals.

One way to expand the reachable set is to check all the left side vertices that are matched to the reachable set one by one, and update the reachable set by taking unions of the current reachable set with the adjacency set of the left side vertex. Since the both sets are type 1 or type 2 intervals, the union operation takes constant time. The union operation needs to be performed no more than $n(i)$ times, where $n(i)$ is the number of matched left side vertices before checking a_i. Checking the reachable set for finding an unmatched vertex can also be done in $O(n(i))$ time. Therefore, checking a_i needs $O(n(i))$ time. Also, note that a_i needs to be checked only if there has not been a left side vertex on the same wavelength of a_i that was checked before and failed to find an unmatched vertex. There are k wavelengths, as a result, in the algorithm, the total time spent in expanding the reachable set of the vertices is $O(k^2)$. The time spent in updating the matching is also $O(k^2)$. Thus, the running time of this algorithm is $O(k^2)$. Plus the sorting, the scheduling takes $O(k^2 + Nk\log(Nk))$ time.

There also exist two left side vertices that maximally expand the reachable set in when the conversion is circular symmetrical. However, these two vertices are not very easy to find. One might still need to scan all the adjacency set of the matched left side vertices and compare them. Thus, finding these two vertices may be more complicated than simply taking the union of all of the adjacency sets one by one. The latter is also much easier and straightforward to implement.

7 Simulation Results

Besides giving proofs and analyses for the proposed scheduling algorithms, we also implemented the algorithms in software and tested them by simulations. We tested the interconnects of two typical sizes, one with eight input fibers and eight output fibers and with eight wavelengths on each fiber, and the other with 16 input fibers and 16 output fibers and with 16 wavelengths on each fiber.

In the simulations, we assume that the arrivals of the connection requests at the input channels are bursty: An input channel alternates between two states, the "busy" state and the "idle" state. When in the "busy" state, it continuously receives connection requests and all the connection requests go to the same destination. When in the "idle" state, it does not receive any connection requests. The length of the busy and idle periods follows geometric distribution. The network performance is measured by the *blocking probability* which is defined as the ratio of the number of rejected connection requests over the number of arrived connection requests. The durations of the connections are one time slot and for each experiment the simulation program was run for 100,000 time slots.

In Fig. 9, we plot the blocking probability of the interconnect as a function of conversion distance of the two types of wavelength conversions when the connection requests do not have priority. We use the maximum matching algorithms to maximize network throughput, or equivalently, to minimize blocking probability. We tested under two traffic loads, $\rho = 0.6$ where average busy period 15 time slots and average idle period 10 time slots, and $\rho = 0.8$ where average busy period 40 time slots and average idle period 10 time slots. We can see that for both types of conversions the blocking probability decreases as the conversion distance increases. But, when the conversion

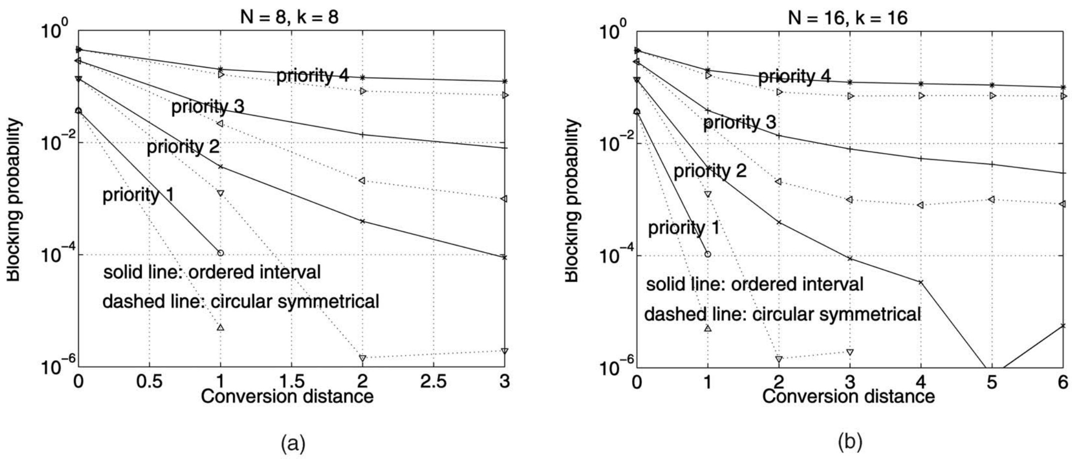


Fig. 10. Blocking probability of WDM interconnects under bursty traffic when the connection requests have priorities. The solid lines are for the ordered interval wavelength conversion and the dashed lines are for the circular symmetrical wavelength conversion. (a) 8×8 interconnect with eight wavelengths per fiber. (b) 16×16 interconnect with 16 wavelengths per fiber.

distance is larger than a certain value, the decease of blocking probability is marginal. In this case, there is little benefit for further increasing the conversion degree, which is exactly the reason for using limited range wavelength converters other than full range wavelength converters. We can also see that the circular symmetrical conversion has smaller blocking probability than the ordered interval conversion because of the extra conversions allowed at the boundaries.

In Fig. 10, we plot the blocking probability when the connection requests have priorities. We use the optimal matching algorithms to maximize network throughput and give service differentiation. The tested traffic load is $\rho = 0.8$, where the average busy period is 40 time slots and average idle period 10 time slots. There are four priorities, with 10, 20, 30, and 40 percent of the total traffic, from the highest priority (priority 1) to the lowest priority (priority 4), respectively. We can see that the optimal matching algorithms achieve good service differentiation. For example, in Fig. 10b, we can see that in a 16×16 interconnect with 16 wavelengths per fiber, for ordered interval wavelength conversion when the conversion distance is 3, the blocking probability of priority 4 is about 10^{-1}, while the blocking probability of priority 2 is about 10^{-4}. The blocking probability of priority 1 should be even smaller, but cannot be seen here because the reliable values of small blocking probability are extremely hard to obtain by simulations and we will use analytical models to find them in our future work.

8 Conclusions

In this paper, we have presented optimal scheduling algorithms to resolve output contentions in bufferless time slotted WDM optical interconnects with limited range wavelength conversion ability. We have introduced the request graph and showed that the problem of maximizing network throughput is equivalent to finding a maximum matching in the request graph. We then gave the First Available Algorithm that runs in $O(k)$ time for finding a maximum matching in the request graph for the ordered interval wavelength conversion, where k is the number of wavelengths per fiber. We also considered optimal scheduling for connection requests with priorities and gave the Downwards Expanding Algorithm that runs in $O(kD + Nk\log(Nk))$ time for finding an optimal matching in a weighted request graph for the ordered interval wavelength conversion, where N is the number of input/output fibers and D is the conversion degree. Finally, we considered the circular symmetrical wavelength conversion scheme and gave optimal scheduling algorithms for nonprioritized scheduling in $O(kD)$ time and prioritized scheduling in $O(k^2 + Nk\log(Nk))$ time. The proposed scheduling algorithms were also evaluated by simulations under bursty traffic. Our future work includes developing analytical models for performance evaluation of the interconnects under these scheduling algorithms.

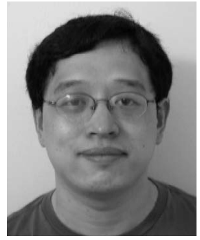
Acknowledgments

This work was supported in part by the US National Science Foundation under grant numbers CCR-0073085 and CCR-0207999.

References

[1] B. Mukherjee, "WDM Optical Communication Networks: Progress and Challenges," *IEEE J. Selected Areas in Comm.,* vol. 18, no. 10, pp. 1810-1824, 2000.
[2] D.K. Hunter, M.C. Chia, and I. Andonovic, "Buffering in Optical Packet Switches," *J. Lightwave Technology,* vol. 16 no. 12, pp. 2081-2094, 1998.
[3] M. Kovacevic and A. Acampora, "Benefits of Wavelength Translation in All-Optical Clear-Channel Networks," *IEEE J. Selected Areas in Comm.,* vol. 14, no. 5, pp. 868-880, 1996.
[4] S.L. Danielsen, C. Joergensen, B. Mikkelsen, and K.E. Stubkjaer, "Analysis of a WDM Packet Switch with Improved Performance under Bursty Traffic Conditions Due to Tunable Wavelength Converters," *J. Lightwave Technology,* vol. 16, no. 5, pp. 729-735, 1998.
[5] N. McKeown, P. Varaiya, and J. Warland, "Scheduling Cells in an Input-Queued Switch," *IEEE Electronics Letters,* pp. 2174-2175, 1993.
[6] N. McKeown, "The iSLIP Scheduling Algorithm Input-Queued Switch," *IEEE/ACM Trans. Networking,* vol. 7, pp. 188-201, 1999.
[7] H.J. Chao, C.H. Lam, and E. Oki, *Broadband Packet Switching Technologies,* first ed. Wiley-Interscience, 2001.

[8] W.J. Goralski, *Optical Networking and WDM,* first ed. McGraw-Hill, 2001.
[9] R. Ramaswami and K.N. Sivarajan, *Optical Networks: A Practical Perspective,* first ed. Academic Press, 2001.
[10] T. Tripathi and K.N. Sivarajan, "Computing Approximate Blocking Probabilities in Wavelength Routed All-Optical Networks with Limited-Range Wavelength Conversion," *IEEE J. Selected Areas in Comm.,* vol. 18, pp. 2123-2129, 2000.
[11] L. Xu, H.G. Perros, and G. Rouskas, "Techniques for Optical Packet Switching and Optical Burst Switching," *IEEE Comm. Magazine,* pp. 136-142, 2001.
[12] R. Ramaswami and G. Sasaki, "Multiwavelength Optical Networks with Limited Wavelength Conversion," *IEEE/ACM Trans. Networking,* vol. 6, pp. 744-754, 1998.
[13] X. Qin and Y. Yang, "Nonblocking WDM Switching Networks with Full and Limited Wavelength Conversion," *IEEE Trans. Comm.,* vol. 50, no. 12, pp. 2032-2041, 2002.
[14] Y. Yang, J. Wang, and C. Qiao, "Nonblocking WDM Multicast Switching Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 12, pp. 1274-1287, 2000.
[15] M.S. Borella and B. Mukherjee, "Efficient Scheduling of Nonuniform Packet Traffic in a WDM/TDM Local Lightwave Network with Arbitrary Transceiver Tuning Latencies," *IEEE J. Selected Areas in Comm.,* vol. 14, no. 5, pp. 923-934, 1996.
[16] G.N. Rouskas and V. Sivaraman, "Packet Scheduling in Broadcast WDM Networks with Arbitrary Transceiver Tuning Latencies," *IEEE/ACM Trans. Networking,* vol. 14, no. 3, pp. 359-370, 1997.
[17] Z. Zhang and Y. Yang, "Distributed Scheduling Algorithms for Wavelength Convertible WDM Optical Interconnects," *Proc. 17th IEEE Int'l Parallel and Distributed Processing Symp.,* 2003.
[18] E.L. Lawler, *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.
[19] J. Hopcroft and R. Karp, "An $n^{\frac{5}{2}}$ Algorithm for Maximum Matchings in Bipartite Graph," *SIAM J. Computing,* vol. 2, no. 4, pp. 225-231, 1973.
[20] F. Glover, "Maximum Matching in Convex Bipartite Graph," *Naval Research Logistics Quarterly,* vol. 14, pp. 313-316, 1967.
[21] W. Lipski Jr. and F.P. Preparata, "Algorithms for Maximum Matchings in Bipartite Graphs," *Naval Research Logistics Quarterly,* vol. 14, pp. 313-316, 1981.
[22] E.L. Lawler, *Introduction to Graph Theory.* Holt, Rinehart and Winston, 1976.
[23] G. Shen et al., "Performance Study on a WDM Packet Switch with Limited-Range Wavelength Converters," *IEEE Comm. Letters,* vol. 5, no. 10, pp. 432-434, 2001.
[24] H. Qin, S. Zhang, and Z Liu, "Dynamic Routing and Wavelength Assignment for Limited-Range Wavelength Conversion," *IEEE Comm. Letters,* vol. 5, no. 3, pp. 136-138, 2003.
[25] X. Masip-Bruin et al., "Routing and Wavelength Assignment under Inaccurate Routing Information in Networks with Sparse and Limited Wavelength Conversion," *Proc. IEEE GLOBECOM Conf. '03,* vol. 5, pp. 2575-2579, 2003.



Zhenghao Zhang received the BEng and MS degrees in electrical engineering from Zhejiang University, People's Republic of China, in 1996 and 1999, respectively. From 1999 to 2001, he worked in industry as a software engineer in embedded systems design. Since 2001, he has been working toward the PhD degree in the Department of Electrical and Computer Engineering at the State University of New York at Stony Brook. His research interest includes scheduling and performance analysis of optical networks. He is a student member of the IEEE and IEEE Computer Society.

Yuanyuan Yang received the BEng and MS degrees in computer science and engineering from Tsinghua University, Beijing, China, and the MSE and PhD degrees in computer science from Johns Hopkins University, Baltimore, Maryland. Dr. Yang is a professor of computer engineering and computer science at the State University of New York at Stony Brook. Dr. Yang's research interests include parallel and distributed computing and systems, high speed networks, optical and wireless networks, and high performance computer architecture. Her research has been supported by the US National Science Foundation (NSF) and US Army Research Office (ARO). She has published extensively in major journals and refereed conference proceedings and holds six US patents in these areas. She is an editor for the *IEEE Transactions on Parallel and Distributed Systems* and the *Journal of Parallel and Distributed Computing*. Dr. Yang has served on NSF review panels and program/organizing committees of numerous international conferences in her areas of research. She is a senior member of the IEEE and the IEEE Computer Society.