INFORMS Journal on Computing
Vol. 8, No. 1, Winter 1996

0899-1499/96/0801-0063 $01.25

Isomorphic Routing on a Toroidal Mesh

WEIZHEN MAO AND DAVID M. NICOL / *Department of Computer Science, The College of William and Mary, Williamsburg, VA 23187-8795; Email: {wm, nicol}@cs.wm.edu*

(Received: March 1993; revised April 1994; accepted April 1995)

We study a routing problem that arises on SIMD parallel architectures whose communication network forms a toroidal mesh. We assume there exists a set of k message descriptors $\{(x_i, y_i) | i = 1, 2, \ldots, k\}$, where (x_i, y_i) indicates that the ith message's recipient is offset from its sender by x_i hops in one mesh dimension, and y_i hops in the other. Every processor has k messages to send, and all processors use the same set of message descriptors. The SIMD constraint implies that at any routing step, every processor is actively routing messages with the same descriptors as any other processor. We call this *Isomorphic Routing*. Our objective is to find the isomorphic routing schedule with the minimum makespan. We consider a number of variations on the problem, yielding complexity results from $O(k)$ to NP-complete. Most of our results follow after we transform the problem into a scheduling problem, where it is related to other well-known scheduling problems.

The issue of routing messages in a parallel computer network has attracted a considerable amount of attention. A host of problem variations exist. For example, some models presume that every processor i holds a number and that one wishes to implement some permutation (e.g., [7]). Another variation is to assume that each processor i has a list of messages each of which is destined for an arbitrary processor, this is known as "all-to-all personalized communication".[4] Our problem is a constrained case of all-to-all personalized communication, on an $n \times m$ toroidal mesh. It is also a constrained case of the general "compiled communication" problem studied in [1], where the problem is to construct a communication schedule for an irregular computation. The problem arises, for instance, with mesh-based calculations using complex discretization templates (e.g., see [8]). Solutions such as ours are appropriately at compile-time or load-time, if the communication pattern can be determined. This is to be contrasted with dynamic off-line algorithms (e.g., see [6]).

To begin with, in our problem, we can always describe a message's destination in terms of the offset in both mesh dimensions X and Y of the source processor. Thus, a pair (x, y) describes a message's routing requirements. Observe however that a message need not travel exactly x units in the X dimension and y in the Y—because of wrap-around, it may equally well choose to travel $m - x$ units in X and/or $n - y$ units in Y. Now imagine a parallel computation where every processor performs the same computation, but on different data. Further suppose that the pattern of messages every processor sends is the same, (e.g., patterns associated with discretization stencils[8]). We may thus describe the communication requirements of the entire computation in terms of the offsets $\{(x_1, y_1), \ldots, (x_k, y_k)\}$ of the k messages a single processor sends. We will say that the $n \times m$ different messages with a common offset pair are all *isomorphic*.

Every processor has four communication ports, referenced as North, East, West, and South (N, E, W, and S). We assume that the communication links are full-duplex. We are interested in SIMD (Single Instruction Multiple Data) architectures, where processors execute the same instruction stream in lock-step. Unless the architecture provides special support for local indirect addressing (which is much slower even when provided), an implication of SIMD processing is that at every instant, the set of messages moving through all ports of a common type (e.g., N) are isomorphic. We desire a routing schedule that minimizes the time required to complete the communication, i.e., the makespan.

We will examine variations of the problem, finding they have a surprising range of complexities. The variations derive from assumptions concerning how many communication ports may be active at a time, and whether a message must be fully routed once it begins moving or if it can be temporarily buffered at an intermediate processor. The assumptions and associated complexities of finding the optimal routing schedules are given below:

- **One port active at a time:** $O(k)$;
- **All ports active, temporary buffering allowed:** $O(k \log k)$;
- **All ports active, no temporary buffering:** NP-complete.

We organize this paper as follows. In Section 1, we give the problem definitions. In Section 2, we study the first and third variations listed above. Section 3 develops an algorithm for the second variation which allows temporary buffering, and Section 4 develops an $O(k \log k)$ implementation of the algorithm. Section 5 presents our conclusions. The Appendix proves some useful lemmas in detail.

1. Problem Definitions

Let us now state the problem more formally. In a toroidal mesh of $n \times m$ processors (with n rows and m columns), a set of messages $M = \{m_1, m_2, \ldots, m_k\}$ is to be sent from each of the $n \times m$ processors (the sources) to some other processors (the destinations). Each message m_i is represented by a pair of integers (x_i, y_i) giving the relative offset of its destination from its source. Assume $0 \le x_i \le m - 1$ and $0 \le y_i \le n - 1$. We wish to design a schedule so that all messages at each

Subject classifications: Analysis of algorithms: computational complexity. Networks/graphs: message routing. Production/scheduling: open shop scheduling.

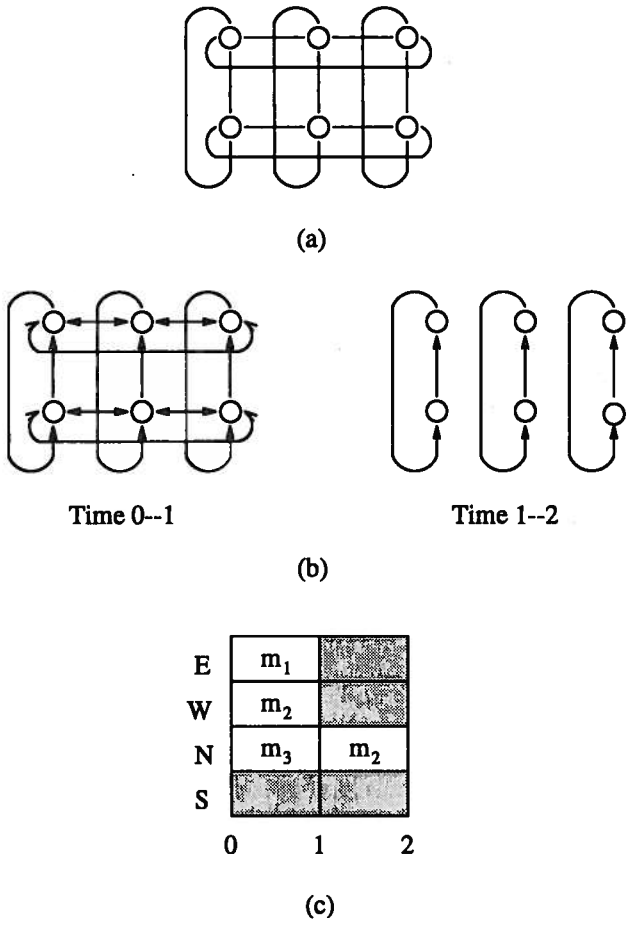

Figure 1. An example.

processor are sent to (and received by) their destinations in the minimum amount of time. Depending on the problem variation, at any time a processor can send one message in one of four directions (N, E, W, or S), or at any time a processor can send up to four messages, one in each direction. We assume it always takes one time unit for a message to traverse one link. We notice that for any message $m_i = (x_i, y_i)$ there are four possible ways to send it, East and North (x_i, y_i), East and South $(x_i, -(n - y_i))$, West and North $(-(m - x_i), y_i)$, and finally, West and South $(-(m - x_i), -(n - y_i))$. Because the mesh is toroidal, they all reach the same destination. Depending on the problem variation, we either assume that a message must be routed to completion in a successive series of steps, or that a message's movement can be fragmented (not continuous), e.g., one step North, two steps buffered, one step West, another step North, and so on.

For example, in a 2×3 toroidal mesh shown in Figure 1(a), 3 messages are to be sent, they are $m_1 = (1, 0)$, $m_2 = (2, 1)$, and $m_3 = (0, 1)$. Assuming that all ports may be active simultaneously, we easily determine that the makespan of the optimal schedule is 2, regardless of whether temporary buffering is allowed or not. From time 0 to time 1, each processor sends m_1 East to its destination, m_2 West, and m_3 North to its destination. From time 1 to time 2, each processor sends m_2 North to its destination. The schedule is illustrated in Figure 1(b). Under our assumption of isomorphic message routing, each processor does exactly the same thing at the same time. Any time a processor sends out a message on one port (e.g., N), in the following time step a message isomorphic to it is received on the opposite port (e.g., S), save that one unit of routing service in one dimension (e.g., Y) has been given. This observation suggests that we can approach the scheduling problem in terms of a single processor giving routing service to each of its k messages. The schedule for one processor can be shown by the traditional Gantt chart as in Figure 1(c).

A message may travel either direction within a dimension. This allows the possibility of schedules that cause a message to "backtrack", e.g., move 3 units West, and later move 4 units East. In the case when temporary buffering is provided at each processor, such a schedule can always be improved (at least not degraded) by removing the backtracking loop, i.e., if C^*_{max} is the minimized makespan for an instance of the isomorphic routing problem, there exists a backtracking-free schedule with cost C^*_{max}. When there is no temporary buffering, backtracking may be needed just to keep a message moving until it reaches its destination. In the remainder we will confine our attention to backtracking-free schedules.

The problem defined above can be converted to an equivalent problem similar to the open shop scheduling. We are given four machines E, W, N, S, in which E, and W are identical in function but give different service times, as do N and S. There are k jobs, $J_1, J_2, \ldots, J_k$. Each job J_i consists of two tasks X_i and Y_i, where X_i can only be executed by E or W but not both (because there is no backtracking), taking x_i or $m - x_i$ time units, respectively, and Y_i can only be executed by N or S but not both, taking y_i or $n - y_i$ time units, respectively. The integers m, n, x_i's and y_i's are as defined in the original problem above. Tasks X_i and Y_i cannot be executed simultaneously at any time, but may be broken into unit-time slices. However, in some problem variations a job J_i may be *suspended* once it has begun execution, a feature that corresponds with a message being buffered en route. Our goal is to find a schedule to execute all jobs so that the makespan or the maximum completion time C_{max} is minimized.

We clarify a few concepts. First, notice the difference between "job" and "task." A job consists of two tasks (or operations), each of which is to be executed by one of the two common function machines. Second, notice the difference between "suspended" and "preempted". The former means that the execution of a *job* can be interrupted, implying that in the execution period of the job there may be a time interval during which no machine is executing any task of the job, while the latter means that the execution of a *task* can be interrupted. Consequently, a schedule in which jobs may be suspended may or may not be a preemptive schedule, and vice versa.

This problem definition suggests a new group of problems, which we call the multi-operation multi-machine

scheduling problems. In the classical multi-operation model,[5] each job requires execution on more than one machine. In an open shop the order in which a job passes through the machines is immaterial, whereas in a flow shop each job has the same machine ordering and in a job shop the jobs may have different machine orderings. In the multi-operation multi-machine model, instead of having just one machine to perform a certain kind of task for a job, there is a back-up machine with the same function and a possibly different cost.

We can distinguish the situations in which a task requires identical service at either common function machine, or has different service requirements that depend on the machine. Our problem is a special case of the latter. In particular, we assume that for each pair of common function machines there exists an integer c ($c = n$ for N and S, $c = m$ for E and W) such that a task with demand x_i requires x_i units on one machine and $c - x_i$ units on the other. In this case we will say that the machines give *complementary* service.

In the remainder we will refer to problem variations by the following names.

P_1: Only one machine (out of all four) may be executing at a time.
P_2: All four machines may execute simultaneously, jobs may be suspended, common function machines give complementary service.
P_3: All four machines may execute simultaneously, jobs may not be suspended, common function machines give complementary service.
P_4: All four machines may execute simultaneously, jobs may be suspended, common function machines give uniform service.
P_5: All four machines may execute simultaneously, jobs may not be suspended, common function machines give uniform service.

P_1, P_2, and P_3 have meaning in the context of the isomorphic routing problem in the sense that P_1 corresponds to the case in which only one port is active at a time, P_2 corresponds to the case in which all ports are active and temporary buffering is allowed, and P_3 corresponds to the case in which all ports are active and temporary buffering is not allowed; P_4 and P_5 are natural variations of the multi-operation multi-machine scheduling problem. We will establish complexity bounds on each of these problems.

2. Complexity results for P_1, P_3, P_4, and P_5

Problem P_1 allows only one machine to be executing at a time. The solution is trivial. Step through the jobs sequentially, giving exhaustive service to one task, and then the other, in each case selecting the machine which serves the task most quickly. $2k$ comparisons are performed in the course of selecting machines, giving the algorithm complexity $O(k)$. We then have the following theorem:

Theorem 1. *P_1 is solvable in time $O(k)$.*

While not very interesting in the scheduling context, the above situation follows from the isomorphic routing problem under the constraint that at any step, only one communication port can be active. This is a seemingly natural constraint, but it is not always required. For example, the Thinking Machines CM-2 is able to communicate on all ports simultaneously.[1] Indeed, the problem studied in [1] is similar to ours, in that it seeks to schedule communication (albeit irregular, as opposed to our isomorphic assumption) on the CM-2's hypercube communication network.

Next we show that P_3, P_4, and P_5 are NP-complete. First consider P_4, where common function machines give uniform service. Assume that machines M_1, M_2 are identical, as are M_3, M_4. There are k jobs, $J_1, J_2, \ldots, J_k$. Each job J_i consists of two tasks X_i and Y_i, where X_i can only be executed by M_1 or M_2 but not both, taking x_i time units on either machine, and Y_i can only be executed by M_3 or M_4 but not both, taking y_i time units on either machine. A job may be suspended, but may never have both its tasks receiving service simultaneously. Our goal is to find a schedule with the minimum makespan C^*_{max}. We shall next prove that whether we allow preemption of tasks or not, the problem is always NP-complete. Note that the NP-completeness of this formulation (an open shop scheduling problem of identical back-up machines with or without preemption) implies the intractability of all general multi-operation multi-machine scheduling problems.

Theorem 2. *P_4 is NP-complete.*

Proof. Consider the corresponding decision problem, in which given a bound B, we are asked whether there is a schedule with makespan $C_{max} \leq B$. This decision problem of P_4 is certainly in NP. Now we show that it can be polynomially reduced from the NP-complete PARTITION.[2] Given any instance of PARTITION, $A = \{a_1, a_2, \ldots, a_k\}$ (positive integers), we construct an instance of the decision problem, in which there are k jobs with $x_i = a_i$ and $y_i = 0$ for $i = 1, 2, \ldots, k$. Let $B = \frac{1}{2} \sum_{i=1}^{k} a_i$. It is trivial to prove that there exists $A' \subseteq A$ such that $\sum_{a_i \in A'} a_i = \frac{1}{2} \sum_{i=1}^{k} a_i$ iff there is a schedule with $C_{max} \leq B$ for the instance defined. Furthermore, the reduction holds regardless of whether preemption of tasks is allowed or not. ■

Now let us consider P_5, in which a job's service must be continuous, i.e., a job can not be suspended, and common function machines give uniform service. The requirement of continuity of execution of jobs does not prohibit the tasks from being broken into slices which are independently scheduled, so long as a job's execution is not interrupted. It is easy to see that the proof of Theorem 2 can be used without any change to prove the NP-completeness of problem P_5 in both cases of preemption and non-preemption (of tasks). Thus we have the additional result:

Theorem 3. *P_5 is NP-complete.*

Now suppose that a job's service must be continuous, and that common function machines give complementary service. If we assume that a task may be broken into unit-time slices, this formulation corresponds directly to an isomorphic routing problem where we require that once begun, a

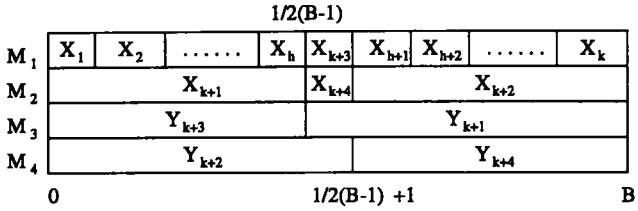

Figure 2. A schedule with $C_{max} = B$ for the instance of the decision problem of P_3.

message continues to move at each step until it reaches its destination. It turns out that this variation is also intractable.

Theorem 4. *P_3 is NP-complete.*

Proof. Consider the corresponding decision problem, in which given a bound B, we are asked whether there is a schedule with makespan $C_{max} \leq B$. This decision problem of P_3 is certainly in NP. Now we show that it can be polynomially reduced from the NP-complete PARTITION. Given any instance of PARTITION, $A = \{a_1, a_2, \ldots, a_k\}$ (positive integers), we construct an instance of the decision problem as follows. Let m and n be two integers larger than $2\sum_{i=1}^{k} a_i + 2$. Given four machines M_1, M_2, M_3, M_4, where M_1 and M_2 are identical in function but give complementary service (x and $m - x$ for workload, x, respectively), as do M_3 and M_4 (y and $n - y$ for workload y, respectively). There are $k + 4$ jobs, $J_1, J_2, \ldots, J_{k+4}$, each of which consists of an X-type task and a Y-type task. Let $x_i = a_i$ and $y_i = 0$ for $i = 1, 2, \ldots, k$, $x_{k+1} = m - \frac{1}{2}\sum_{i=1}^{k} a_i$ and $y_{k+1} = \frac{1}{2}\sum_{i=1}^{k} a_i + 1$, $x_{k+2} = m - \frac{1}{2}\sum_{i=1}^{k} a_i$ and $y_{k+2} = n - \frac{1}{2}\sum_{i=1}^{k} a_i - 1$, $x_{k+3} = 1$ and $y_{k+3} = \frac{1}{2}\sum_{i=1}^{k} a_i$, $x_{k+4} = m - 1$ and $y_{k+4} = n - \frac{1}{2}\sum_{i=1}^{k} a_i$. Finally, let $B = \sum_{i=1}^{k} a_i + 1$. We claim that there exists $A' \subseteq A$ such that $\sum_{a_i \in A'} a_i = \frac{1}{2}\sum_{i=1}^{k} a_i$ iff there is a schedule with $C_{max} \leq B$ for the instance defined.

If there exists $A' \subseteq A$ such that $\sum_{a_i \in A'} a_i = \frac{1}{2}\sum_{i=1}^{k} a_i$ (for notational simplicity, assume $A' = \{a_1, a_2, \ldots, a_h\}$), then we can construct a schedule with $C_{max} = B$ as shown in Figure 2. As a matter of fact, the schedule in Figure 2 is the best possible since for any feasible schedule $C_{max} \geq \lceil \frac{1}{4} \sum_{i=1}^{k+4} (\min\{x_i, m - x_i\} + \min\{y_i, n - y_i\}) \rceil = \sum_{i=1}^{k} a_i + 1 = B$. Notice that the execution of each job must be continuous in the schedule.

If there exists a schedule with $C_{max} = B$, then $X_1, X_2, \ldots, X_k$ and X_{k+3} must be scheduled on M_1, $X_{k+1}, X_{k+2}, X_{k+4}$ on M_2, Y_{k+1}, Y_{k+3} on M_3, and Y_{k+2}, Y_{k+4} on M_4. Since X_{k+1} and Y_{k+1} can not be executed simultaneously, M_2 must execute X_{k+1} at the same time M_3 executes Y_{k+3}. So we say that the executions of X_{k+1} and Y_{k+3} are completely parallel. Since the executions of X_{k+3} and Y_{k+3} are continuous, so are the executions of X_{k+3} and X_{k+1}. Similarly, we can show that the executions of X_{k+4} and X_{k+2} are also continuous. How can the schedule have X_{k+3} on M_1 and X_{k+1}, X_{k+2}, and X_{k+4} on M_2 such that the continuity of X_{k+3} and X_{k+1} and the continuity of X_{k+4} and X_{k+2} are both respected? It is not hard to see that X_{k+3} must be scheduled from time $\frac{1}{2}\sum_{i=1}^{k} a_i$ to time $\frac{1}{2}\sum_{i=1}^{k} a_i + 1$. Therefore set $\{X_1, X_2, \ldots, X_k\}$ is divided into two sets of equal sum. So there exists $A' \subseteq A$ such that $\sum_{a_i \in A'} a_i = \frac{1}{2}\sum_{i=1}^{k} a_i$. ■

3. An Algorithm for P_2

We are now left with the problem of analyzing P_2. This will require most of the remainder of the paper. In P_2, we are given four machines. M_1 and M_2 have common function but give complementary service, so as M_3 and M_4. There are k jobs, $J_1, J_2, \ldots, J_k$. Each job J_i consists of two tasks X_i and Y_i, where X_i can only be executed by M_1 or M_2 but not both, taking x_i or $m - x_i$ time units, respectively, and Y_i can only be executed by M_3 or M_4 but not both, taking y_i or $n - y_i$ time units, respectively. All parameters are integers. Jobs may be suspended, i.e., the execution of a job may not be continuous, and tasks can be preempted at integral points, i.e., they can be broken into unit-time slices to execute. We wish to find a schedule of all jobs with the minimum makespan. P_2 is in fact the isomorphic routing problem in which all ports are active and temporary buffering is allowed. The complexity result is in our last theorem:

Theorem 5. *P_2 is solvable in time $O(k \log k)$.*

Our approach will be to recognize that P_2 is a variation on a scheduling problem, denoted by P_2', where the decision of which of the two common function machines to use for any given task is a given input parameter; the sign of x_i or y_i determines which machine to use. In the isomorphic routing problem this is equivalent to specifying the specific directions the message must travel. This might arise, for instance, if the N and E ports could be used only for sending messages, whereas the S and W ports could be used only for receiving them.

Assuming that the machine usage is pre-specified, the resulting problem P_2' is related to a paper by Gonzalez and Sahni.[3] The paper studies the general open shop scheduling with preemption, which is defined as follows. There are m machines and n jobs. Each job consists of m tasks (some may have zero execution time) that need to be executed on the m machines correspondingly. No job can have more than one of its tasks executed simultaneously, but the execution of any task may be preempted. The goal is to construct a schedule with the minimum makespan. Gonzalez and Sahni proved that the minimum makespan $C^*_{max} = \alpha \equiv \max_{i,j}\{L_i, T_j\}$, where L_i is the sum of execution times of all tasks of job J_i, and T_j is the sum of execution times of all tasks that need to be executed on machine M_j. We notice that α is solely determined by the parameters of the problem and is an obvious lower bound to the makespan of any schedule. To construct the schedule with makespan α, i.e., the optimal schedule, for any instance with m machines, n jobs, and r nonzero tasks, an $O(r(\min\{r, m^2\} + m \log n))$ algorithm is presented in the paper. There is an interesting property about this algorithm that we shall use in the discussion below. That is, when all parameters of the problem are integers, preemptions of tasks always occur at integral points in the optimal schedule.

It is easy to see that P_2' is in fact a restricted variation of the open shop scheduling problem, in which the number of

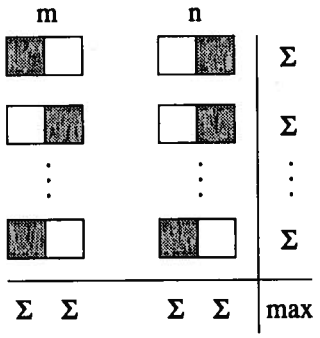

Figure 3. A graphical definition of P_2''.

machines is four, each job has at most two nonzero tasks, all parameters are integers, and preemptions are only allowed at integral points. Furthermore, we also notice that the minimum makespan for any instance of P_2', C^*_{max}, is at least $\alpha = \max_{i,j}\{L_i, T_j\} = \max\{\max_i\{|x_i| + |y_i|\}, \sum_{\forall x_i>0} x_i, \sum_{\forall x_i<0}(-x_i), \sum_{\forall y_i>0} y_i, \sum_{\forall y_i<0}(-y_i)\}$. When we apply Gonzalez and Sahni's algorithm to P_2', we have an optimal preemptive schedule with $C^*_{max} = \alpha$. Since all preemptions occur at integral points, this is actually the optimal solution to P_2'. The time complexity of Gonzalez and Sahni's algorithm when applied to P_2' is $O(k \log k)$.

In view of this result, our approach will be to take a problem instance of P_2, and determine the task-to-machine assignment that minimizes the corresponding α. Gonzalez and Sahni's algorithm may then be applied to construct the optimal schedule for P_2. Therefore, P_2 is reduced to the problem of finding the optimal task-to-machine assignment. In the remainder of this section we develop an algorithm that makes the needed assignment.

We abstract our problem as P_2''. We are given two sets of items, $X = \{X_1, X_2, \ldots, X_k\}$, and $Y = \{Y_1, Y_2, \ldots, Y_k\}$, and nonnegative integers $x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_k, m$, and n, where $x_i \le m - 1$ and $y_i \le n - 1$ for all i's. We must define an assignment function f^*: $X \cup Y \to \mathbb{N}$ with $f^*(X_i) = x_i$ or $m - x_i$, and $f^*(Y_i) = y_i$ or $n - y_i$ such that $\alpha = \max_{i,j}\{L_i, T_j\} = \max\{\alpha_1, \alpha_2, \alpha_3\}$ is minimized, where

$$\alpha_1 = \max_{\forall i}\{f^*(X_i) + f^*(Y_i)\},$$

$$\alpha_2 = \max\left\{\sum_{\forall i(f^*(X_i)=x_i)} x_i, \sum_{\forall i(f^*(X_i)=m-x_i)} (m - x_i)\right\},$$

$$\alpha_3 = \max\left\{\sum_{\forall i(f^*(Y_i)=y_i)} y_i, \sum_{\forall i(f^*(Y_i)=n-y_i)} (n - y_i)\right\}.$$

We call f^* the *optimal assignment function*. Its corresponding α value is denoted as α^*. The minimum values of α_1, α_2 and α_3 are denoted as α_1^*, α_2^*, and α_3^*, respectively. Note that α^* may not be equal to $\bar{\alpha} \equiv \max\{\alpha_1^*, \alpha_2^*, \alpha_3^*\}$.

The problem can be depicted by Figure 3. There are two columns of integer pairs, each represented by a rectangle consisting of two adjacent squares. The pairs in the left column are m-complementary, meaning that the sum of the two nonnegative integers in a pair is m, while the pairs in the right column are n-complementary. Choose one number in each pair, denoted by shaded squares, and compute row-wise and column-wise sums of the numbers chosen. The cost function is defined to be the maximum of all sums. Then P_2'' is to choose one number in each pair so that the corresponding value of the cost function is minimized.

We first describe an algorithm **A** that defines an assignment function f: $X \cup Y \to \mathbb{N}$ with $f(X_i) = x_i$ or $m - x_i$, and $f(Y_i) = y_i$ or $n - y_i$ such that the resulting α_2 and α_3 are both minimized. Following this, we look at how f^* may deviate from f, and show how to modify f so as to create f^*.

Algorithm A

1. Sort $x_1, x_2, \ldots, x_k$ and $y_1, y_2, \ldots, y_k$ in nondecreasing order, separately.
2. Do the following to each sorted list. The pseudo-code below defines f_X: $X \to \mathbb{N}$ with $f_X(X_i) = x_i$ or $m - x_i$ such that the resulting α_2 is minimized. To define f_Y: $Y \to \mathbb{N}$ for which α_3 is minimized, we simply replace the notations for the X-list by the corresponding notations for the Y-list.

   For notational simplicity, assume $x_1, x_2, \ldots, x_k$ are in nondecreasing order;

   ```
   α2+ ← 0;  α2− ← 0;  i ← 1;  j ← k;
   while  i ≤ j  do
     if  α2+ + xi ≤ α2− + (m − xj)
     then  {fX(Xi) ← xi; α2+ ← α2+ + xi; i ++}
     else  {fX(Xj) ← m − xj; α2− ← α2− + (m − xj); j −−};
   α2* ← max{α2+, α2−};
   ```

3. f: $X \cup Y \to \mathbb{N}$ is the combination of f_X and f_Y.

Given the sorted ordering of the x_i's, the algorithm finds a *turning point* t, where $f(X_i) = x_i$ for $i \le t$, and $f(X_i) = m - x_i$ for $i > t$; furthermore, among all such turning points the one chosen by algorithm **A** minimizes $\max\{\alpha_2^+, \alpha_2^-\}$. That this algorithm defines α_2^* follows from the fact that the optimal schedule must have this structure, for suppose not. Assume there are p and q with $1 \le p < q \le k$ such that $f(X_p) = m - x_p$, and $f(X_q) = x_q$. Since $x_p \le x_q$ and $m - x_p \ge m - x_q$, it follows that $\max\{m - x_p, x_q\} \ge \max\{x_p, m - x_q\}$, so that changing the assignment for X_p and X_q does not increase the corresponding α_2. We may apply this argument repeatedly until the resulting assignment exhibits a turning point, as claimed.

Figure 4 shows an example of using algorithm **A** to compute α_2^*. The numbers in the circles are the values of function f_X of the corresponding tasks, which are also called *choices*. From now on, we shall use the diagrams similar to Figure 4 to represent the definition of f. They are called the *assignment diagrams*.

We see from the above discussion that α_2^* and α_3^* can be obtained by algorithm **A** in time $O(k \log k)$, while α_1^* can be obtained by choosing $\min\{x_i, m - x_i\}$ for task X_i and $\min\{y_i, n - y\}$ for task Y_i. However, the difficulty we face is that these optimalities may not be achieved at the same time, i.e., the assignment minimizing α_2 and α_3 may not be consistent with the assignment minimizing α_1. To highlight the

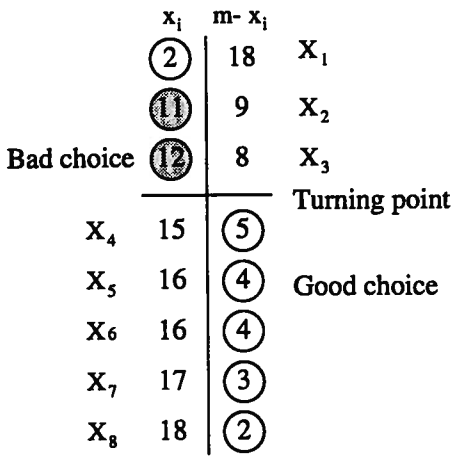

Figure 4. An example of using algorithm **A** to compute α_2^*.

differences we will say that $f(X_i)$ (alt., $f(Y_i)$) is a *bad choice* if $f(X_i) \neq \min\{x_i, m - x_i\}$ (alt., $f(Y_i) \neq \min\{y_i, n - y_i\}$), and that $f(X_i)$ (alt., $f(Y_i)$) is a *disastrous choice* if $f(X_i) \neq \min\{x_i, m - x_i\}$ (alt., $f(Y_i) \neq \min\{y_i, n - y_i\}$) and $f(X_i) + f(Y_i) > \alpha$. In the example in Figure 4, the shaded circles represent the bad choices. It is easy to see that bad choices, if exist, always form a contiguous block. Without loss of generality, assume that the block of bad choices is in the (upper) left column of the assignment diagram. Therefore, the block ends at the turning point. We observe that if f contains no disastrous choices, then $f^* = f$ and $\alpha^* = \alpha$. Should f contain disastrous choices, we need to consider modifying it in order to find the function f^* that minimizes α.

Let us assume then that we have obtained f by applying algorithm **A** to the X list (and so have found the X assignment function f_X), and to the Y list (and so have found the Y assignment function f_Y), and have identified at least one disastrous choice. We have developed a number of results that help us to identify tasks whose assignment in f can differ from their assignment in f^*. Most importantly, these results severely constrain the number of such tasks.

We proceed now by making some definitions, and stating certain results founded upon them. Without loss of generality, we assume $m > n$ in the remainder. Given f, which is the combination of f_X and f_Y, we define the following notations.

- B_X (alt., B_Y): The set of bad choices in f_X (alt., f_Y).
- D_X (alt., D_Y): The set of disastrous choices in f_X (alt., f_Y).
- L_X (alt., L_Y): The set of left-column choices in f_X (alt., f_Y).
- R_X (alt., R_Y): The set of right-column choices in f_X (alt., f_Y).
- U_X (alt., U_Y): A subset of L_X (alt., L_Y).
- V_X (alt., V_Y): A subset of R_X (alt., R_Y).
- S: $U_X \cup V_X \cup U_Y \cup V_Y$.
- $\alpha_1(S)$: The corresponding α_1 of the assignment obtained from modifying f by making alternative choices, also called *switches*, of those in S.
- $t_i(S)$: The term contributed to $\alpha_1(S)$ by tasks X_i and Y_i, i.e., $t_i(S)$ is $f(X_i) + f(Y_i)$ if $f(X_i), f(Y_i) \notin S$; it is $m - f(X_i) + f(Y_i)$ if $f(X_i) \in S$ and $f(Y_i) \notin S$; it is $f(X_i) + n - f(Y_i)$ if $f(X_i) \notin S$ and $f(Y_i) \in S$; and it is $m - f(X_i) + n - f(X_i)$ if $f(X_i), f(Y_i) \in S$.
- $\alpha_2(S)$ (alt., $\alpha_3(S)$): The corresponding α_2 (alt., α_3) of the assignment obtained from modifying f_X (alt., f_Y) by switching the choices in S. Note that only U_X, V_X (alt., U_Y, V_Y) affect $\alpha_2(S)$ (alt., $\alpha_3(S)$).
- $\alpha(S)$: $\max\{\alpha_1(S), \alpha_2(S), \alpha_3(S)\}$, i.e., the corresponding α of the assignment obtained from modifying f by switching the choices in S.

By definitions above, we know that any set of switches S, which is specified by U_X, V_X, U_Y, V_Y, determines an assignment different from f and vice versa. Therefore, the optimal assignment function f^* is also defined by a set of switches S^* (may not be unique). To determine S^*, we may have to check all possible combinations of U_X, V_X, U_Y, V_Y (it turns out later that only a limited number of combinations need to be considered). We need one more definition. Given any $S = U_X \cup V_X \cup U_Y \cup V_Y$, we say that $f(X_i)$ (alt., $f(Y_i)$) is a *potential switch* if either $f(X_i)$ (alt., $f(Y_i)$) is a disastrous choice, or $f(X_i)$ (alt., $f(Y_i)$) is a bad choice, $f(Y_i)$ (alt., $f(X_i)$) is in V_Y (alt., V_X), and $f(X_i) + n - f(Y_i) > \alpha_3(S)$ (alt., $m - f(X_i) + f(Y_i) > \alpha_2(S)$). The following four lemmas serve to constrain the number of possible switches we must consider. Their proofs are relegated to the Appendix.

Lemma 1. *If* $|B_X| \geq 3$, *then* $f^* = f$.

Lemma 2. $|D_Y| \leq 2$.

Lemma 3. *In* $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$, $|U_X^*| \geq |V_X^*|$ *and* $|U_Y^*| \geq |V_Y^*|$.

Lemma 4. *In* $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$, *all members of* U_X^* *and* U_Y^* *are potential switches.*

Now consider the implications of these results, i.e., how they can be used to reduce the numbers of sets of switches $S = U_X \cup V_X \cup U_Y \cup V_Y$ that need to be considered in order to determine the set of optimal switches $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$ for f^*. By Lemma 1, we only have to worry about situations when $|B_X| \leq 2$. By Lemma 4, we know that U_X^* can only contain potential switches, which are recognizable bad choices. Thus, $U_X^* \subseteq B_X$ and $|U_X^*| \leq |B_X| \leq 2$. Consequently, there are at most $O(\sum_{j=0}^{2} \binom{2}{j}) = O(1)$ candidates for U_X^*. By Lemma 3, $|V_X^*| \leq |U_X^*| \leq 2$, and $V_X^* \subseteq R_X$, $|R_X| \leq k$, hence there are at most $O(\sum_{j=0}^{2} \binom{k}{j}) = O(k^2)$ candidates for V_X^* once U_X^* is chosen. By Lemma 4, U_Y^* can only contain potential switches, which are disastrous choices in f_Y (there are at most 2 by Lemma 2) or bad choices $f(Y_i)$ with $f(X_i) \in V_X^*$. So $|U_Y^*| \leq |D_Y| + |V_X^*| \leq 2 + 2 = 4$. Therefore, there are at most $O(\sum_{j=0}^{4} \binom{4}{j}) = O(1)$ candidates for U_Y^* once U_X^*, V_X^* are chosen. By Lemma 3, $|V_Y^*| \leq |U_Y^*| \leq 4$, and $V_Y^* \subseteq R_Y$, $|R_Y| \leq k$, hence there are at most $O(\sum_{j=0}^{4} \binom{k}{j}) = O(k^4)$ candidates for V_Y^* once U_X^*, V_X^*, U_Y^* are chosen. Considering all $O(1) \cdot O(k^2) \cdot O(1) \cdot O(k^4) = O(k^6)$ combinations, we can determine S^* in time $O(k^6)$.

We describe this algorithm for problem P_2 formally as follows.

1. If $m < n$, rotate the mesh by 90 degrees, and redefine the parameters in the new coordinate system.
2. Use algorithm **A** twice to define f which minimizes $\max\{\alpha_2, \alpha_3\}$.
3. If the block of bad choices in an assignment diagram is not in the left column, rotate the diagram by 180 degrees, and exchange the roles of the two machines involved in the assignment.
4. If there are no disastrous choices in both f_X and f_Y, let f^* be f, α^* be $\bar{\alpha}$, and go to step 6. Otherwise continue in step 5.
5. List all possible definitions for S^*. For each, compute its α. Let f^* be the function determined by the set of switches which results in the smallest α (α^*).
6. Use Gonzalez and Sahni's algorithm to construct the schedule with $C_{max} = \alpha^*$.

In this algorithm, steps 1, 3, and 4 each take $O(k)$ time, while steps 2 and 6 each take $O(k \log k)$ time. We also know that for step 5, even if we use the brute-force method of checking all possible definitions for S^*, with the help of lemmas 1, 2, 3, 4, the time needed is $O(k^6)$. In the next section, we shall show that step 5 can in fact be implemented in time $O(k \log k)$, thus yielding an $O(k \log k)$ algorithm for P_2 and proving Theorem 5.

4. An $O(k \log k)$ Implementation of the Algorithm

The previous section demonstrated that the routing problem has polynomial complexity and that step 5 in the algorithm is the most costly. Using the four lemmas developed, we showed that step 5 can be implemented in time $O(k^6)$. In this section, we further reduce the number of sets of switches $S = U_X \cup V_X \cup U_Y \cup V_Y$ we must check to $O(k \log k)$, thus completing the proof of Theorem 5. To do so, we need some more lemmas. Their proofs can be found in the Appendix.

Lemma 5. *In $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$, there are no tasks X_i and Y_i such that $f(X_i) \in V_Y^*$ and $f(Y_i) \in V_X^*$.*

Lemma 6. *Suppose $|B_X| = 1$. Then in $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$, $|V_X^*| \le |U_X^*| \le 1$ and $|V_Y^*| \le |U_Y^*| \le 2$. Furthermore, if $D_Y = \{f(Y_1), f(Y_2)\}$, then $\bar{\alpha} < f(X_1) + f(X_2)$ and one of $f(X_1)$ and $f(X_2)$ is the only bad and disastrous choice in f_X.*

Lemma 7. *Suppose $|B_X| = 2$. Then $|D_X| \le 1$ and $|D_Y| \le 1$. Furthermore, if $D_X = \{f(x_1)\}$, then $D_Y = \{f(Y_1)\}$ and in $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$, $|U_X^*| = |U_Y^*| = 1$; if $D_Y = \{fY_1)\}$ and $f(X_1) \notin B_X$, then $f(X_1) \in R_X$.*

We categorize all sets of switches $S = U_X \cup V_X \cup U_Y \cup V_Y$ into three patterns: $S_1 = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_X = \emptyset$, $S_2 = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_Y = \emptyset$, and $S_3 = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_X, V_Y \neq \emptyset$. For each pattern, we want to determine the set of switches that yields the least-cost assignment, namely, S_1^*, S_2^*, and S_3^*. The set of optimal switches S^* is therefore S_i^*, where $i \in \{1, 2, 3\}$ and $\alpha(S_i^*) = \min\{\alpha(S_1^*), \alpha(S_2^*), \alpha(S_3^*)\}$. Lemma 3 and Lemma 4 suggest that we only need to consider $S = U_X \cup V_X \cup U_Y \cup V_Y$, where $|U_X| \ge |V_X|$, $|U_Y| \ge |V_Y|$, and all members of U_X and U_Y are potential switches. By Lemma 1, we assume $|B_X| \le 2$.

Step 1. Determine S_1^* among $S_1 = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_X = \emptyset$.

By lemmas 2, 3, 4, $|V_Y| \le |U_Y| \le |D_Y| + |V_X| = |D_Y| \le 2$. We define two subpatterns: $S_{11} = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_X = \emptyset$ and $|V_Y| \le 1$, and $S_{12} = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_X = \emptyset$ and $|V_Y| = 2$. Let S_{11}^* and S_{12}^* be the sets of optimal switches among S_{11} and S_{12}, respectively. Therefore, the set of optimal switches S_1^* among S_1 is S_{1i}^*, where $i \in \{1, 2\}$ and $\alpha(S_{1i}^*) = \min\{\alpha(S_{11}^*), \alpha(S_{12}^*)\}$.

Step 1.1. Determine S_{11}^* among $S_{11} = U_X \cup V_X \cup U_Y \cup V_Y = \emptyset$ and $|V_Y| \le 1$.

Assume $S_{11}^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$ with $V_X^* = \emptyset$ and $|V_Y^*| \le 1$. By Lemma 4, $U_X^* \subseteq B_X$ and $|U_X^*| \le |B_X| \le 2$. Since $V_X^* = \emptyset$ then by Lemma 4 and the definition of potential switches, only disastrous choices can be members of U_Y, so $U_Y \subseteq D_Y$ and $|U_Y| \le |D_Y| \le 2$. Obviously, for each of the U_X^*, V_X^*, U_Y^* there are $O(1)$ candidates. For V_Y^*, we have $|V_Y^*| \le 1$, $V_Y \subseteq R_Y$, and $|R_Y| \le k$, so there are $O(k)$ candidates for V_Y^*. Hence the time complexity to determine S_{11}^* among $S_{11} = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_X = \emptyset$ and $|V_Y| \le 1$ is $O(1) \cdot O(1) \cdot O(1) \cdot O(k) = O(k)$.

Step 1.2. Determine S_{12}^* among $S_{12} = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_X = \emptyset$ and $|V_Y| = 2$.

Assume $S_{12}^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$ with $V_X^* = \emptyset$ and $|V_Y^*| = 2$. By the same argument as in step 1.1, there are $O(1)$ candidates for each of U_X^*, V_X^*, U_Y^*. Because $2 = |V_Y^*| \le |U_Y^*| \le |D_Y| \le 2$ and $U_Y^* \subseteq D_Y$, U_Y^* must be D_Y and D_Y must contain two choices. Assume $D_Y = \{f(Y_1), f(Y_2)\}$, then $U_Y^* = \{f(Y_1), f(Y_2)\}$. We notice that for any $S_{12} = U_X \cup V_X \cup U_Y \cup V_Y$, where $U_X \subseteq B_X$, $V_X = \emptyset$, and $U_Y = \{f(Y_1), f(Y_2)\}$, for each fixed $f(Y_i) \in V_Y$, including different $f(Y_j) \in R_Y$ in V_Y only affects $\alpha_3(S_{12})$ and $t_j(S_{12})$. Therefore, we want to choose $f(Y_j) \in R_Y$ for each fixed $f(Y_i) \in R_Y$ to minimize $\max\{\alpha_3(S_{12}), t_j(S_{12})\}$. Set $\{f(Y_i), f(Y_j)\}$ is then a candidate for V_Y^*. Procedure **B** finds all such candidates for V_Y^* and then determines S_{12}^*.

Procedure B

1. $S_{12}^* \leftarrow \emptyset \cup \emptyset \cup \emptyset \cup \emptyset$;
2. For each combination of U_X, V_X, U_Y, where $U_X \subseteq B_X$, $V_X = \emptyset$, and $U_Y = \{f(Y_1), f(Y_2)\}$, do
   (a) Create a list L of all choices in R_Y, and for each $f(Y_j) \in L$, define $key(j) = t_j(S_{12})$, where $S_{12} = U_X \cup V_X \cup U_Y \cup V_Y$ with $f(Y_j) \in V_Y$;
   (b) Sort L by nondecreasing values of $key(j)$. We call the resulting list L';
   (c) In L', if there is $f(Y_j)$ such that $f(Y_j) \le f(Y_i)$, where $f(Y_i)$ is the left neighbor of $f(Y_j)$ in L', then delete $f(Y_j)$ from L'. We call the resulting list L'';
   (d) For each $f(Y_i) \in R_Y$, do
   Perform a binary search in L'' to locate the $f(Y_j)$ with the minimum $\max\{\alpha_3(S_{12}), key(j)\}$, where $S_{12} = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_Y = \{f(Y_i), f(Y_j)\}$;
   $V_Y \leftarrow \{f(Y_i), f(Y_j)\}$;
   $S_{12} \leftarrow U_X \cup V_X \cup U_Y \cup V_Y$;
   If $\alpha(S_{12}) < \alpha(S_{12}^*)$ then $S_{12}^* \leftarrow S_{12}$.

In (c) of procedure **B**, we delete $f(Y_j)$ from L' if $f(Y_j)$ is no larger than its left neighbor $f(Y_l)$ in the list. This is because such $f(Y_j)$ when included in V_Y never constitutes S^*_{12}. Let $V_Y = \{f(Y_i), f(Y_j)\}$ and $V'_Y = \{f(Y_i), f(Y_l)\}$. Let $S_{12} = U_X \cup V_X \cup U_Y \cup V_Y$ and $S'_{12} = U_X \cup V_X \cup U_Y \cup V'_Y$. Since $f(Y_l)$ is $f(Y_j)$'s left neighbor, $key(l) \le key(j)$, which implies $t_l(S'_{12}) \le t_j(S_{12})$, and hence $\alpha_1(S'_{12}) \le \alpha_1(S_{12})$. Since $f(Y_l) \ge f(Y_j)$, $\alpha_3(S'_{12}) = \alpha^*_3 + 2n - f(Y_1) - f(Y_2) - f(Y_i) - f(Y_l) \le \alpha^*_3 + 2n - f(Y_1) - f(Y_2) - f(Y_i) - f(Y_j) = \alpha_3(S_{12})$. Furthermore $\alpha_2(S'_{12}) = \alpha_2(S_{12})$. Therefore, $\alpha(S'_{12}) \le \alpha(S_{12})$. For any fixed $f(Y_i) \in R_Y$, the choices $f(Y_j)$ in L'' are sorted by nondecreasing $key(j)$ and by nonincreasing $\alpha^*_3 + 2n - f(Y_1) - f(Y_2) - f(Y_i) - f(Y_j)$. This makes the binary search in (d) applicable. It is obvious that the loop in line 2 is iterated $O(1)$ times, both (b) and (d) take $O(k \log k)$, and the remainings take $O(k)$. So the time complexity of procedure **B** is $O(k \log k)$.

Step 2. Determine S^*_2 among $S_2 = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_Y = \emptyset$.

Assume $S^*_2 = U^*_X \cup V^*_X \cup U^*_Y \cup V^*_Y$ with $V^*_Y = \emptyset$. By Lemma 3 and Lemma 4, $V^*_X \subseteq U^*_X \subseteq B_X$ and $|V^*_X| \le |U^*_X| \le |B_X| \le 2$. Since $V_Y = \emptyset$, by Lemma 4 and the definition of potential switches, we have $U_X \subseteq D_X$. Suppose $|V^*_X| = 2$, then $|U^*_X| = |B_X| = |D_X| = 2$. This is impossible since by Lemma 7, we have $|D_X| \le 1$ when $|B_X| = 2$. So $|V^*_X| \le 1$ and there are $O(k)$ candidates for V^*_X. Once V^*_X is chosen, there are $O(1)$ candidates for each of U^*_X, U^*_Y and V^*_Y. Therefore, S^*_2 can be determined in time $O(k)$.

Step 3. Determine S^*_3 among $S_3 = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_X, V_Y \ne \emptyset$.

Since $V_X \ne \emptyset$, then $U_X \ne \emptyset$ and $B_X \ne \emptyset$. So we will consider two cases: $|B_X| = 1$ and $|B_X| = 2$. If $|B_X| = 1$, go to Step 3.1; if $|B_X| = 2$, go to Step 3.2.

Step 3.1. Determine S^*_3 for the case of $|B_X| = 1$.

By Lemma 6, $|V_Y| \le 2$. We define two subpatterns: $S_{31} = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_X, V_Y \ne \emptyset$ and $|V_Y| = 1$, and $S_{32} = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_X, V_Y \ne \emptyset$ and $|V_Y| = 2$. Let S^*_{31} and S^*_{32} be the sets of optimal switches among S_{31} and S_{32}, respectively. Therefore, the set of optimal switches S^*_3 among S_3 is S^*_{3i}, where $i \in \{1, 2\}$ and $\alpha(S^*_{3i}) = \min\{\alpha(S^*_{31}), \alpha(S^*_{32})\}$.

Step 3.1.1. Determine S^*_{31} among $S_{31} = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_X, V_Y \ne \emptyset$ and $|V_Y| = 1$.

Assume $S^*_{31} = U^*_X \cup V^*_X \cup U^*_Y \cup V^*_Y$ with $V^*_X, V^*_Y \ne \emptyset$ and $|V^*_Y| = 1$. Since $V^*_X \ne \emptyset$ and $|B_X| = 1$, then $U^*_X = B_X$. By Lemma 6 and Lemma 4, we also have $|V^*_X| = 1$ and $U^*_Y \subseteq D_Y \cup \{f(Y_j)\}$, where $f(X_i) \in V^*_X$. Procedure **C** below, a variation of procedure **B**, determines S^*_{31} in $O(k \log k)$ time.

Procedure C

1. $S^*_{31} \leftarrow \emptyset \cup \emptyset \cup \emptyset \cup \emptyset$; $U_X \leftarrow B_X$;
2. Create a list L of all choices in R_Y, and for each $f(Y_j) \in L$, define $key(j) = t_j(S_{31})$, where $S_{31} = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_Y = \{f(Y_j)\}$ and $f(X_j) \notin V_X$ (by Lemma 5);
3. Sort L by nondecreasing values of $key(j)$. We call the resulting list L';
4. In L', if there is $f(Y_j)$ such that $f(Y_j) \le f(Y_l)$, where $f(Y_l)$ is the left neighbor of $f(Y_j)$ in L', then delete $f(Y_j)$ from L'. We call the resulting list L'';
5. For each combination of U_X, V_X, U_Y, where $U_X = B_X$, $V_X = \{f(X_i) | \forall f(X_i) \in R_X$, and $U_Y \subseteq D_Y \cup \{f(Y_i)\}$, do
   Perform a binary search in L'' to locate the $f(Y_j)$ $(j \ne i)$ with the minimum $\max\{\alpha_3(S_{31}), key(j)\}$, where $S_{31} = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_Y = \{f(Y_j)\}$;
   $V_Y \leftarrow \{f(Y_j)\}$;
   $S_{31} \leftarrow U_X \cup V_X \cup U_Y \cup V_Y$;
   If $\alpha(S_{31}) < \alpha(S^*_{31})$ then $S^*_{31} \leftarrow S_{31}$.

Note that in line 2, $t_j(S_{31})$ can be computed although S_{31} is not completely defined (we only know $U_X = B_X$). Since $f(Y_j) \in V_Y$, $f(X_j) \notin V_X$ from Lemma 5. Therefore, $t_j(S_{31}) = f(X_j) + n - f(Y_j)$ if $f(X_j) \notin U_X$; and $t_j(S_{31}) = m - f(X_j) + n - f(Y_j)$ if $f(X_j) \in U_X$. It is clear that lines 1–4 need $O(k \log k)$ time. The for loop in line 5 has $O(k)$ iterations, each taking $O(\log k)$ time. So the time complexity of procedure **C** is $O(k \log k)$.

Step 3.1.2. Determine S^*_{32} among $S_{32} = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_X, V_Y \ne \emptyset$ and $|V_Y| = 2$.

Assume $S^*_{32} = U^*_X \cup V^*_X \cup U^*_Y \cup V^*_Y$ with $V^*_X, V^*_Y \ne \emptyset$ and $|V^*_Y| = 2$. By Lemma 6, $|U^*_X| = |V^*_X| = 1$ and $|U^*_Y| = |V^*_Y| = 2$. So $U^*_X = B_X$. We notice that for any $S_{32} = U_X \cup V_X \cup U_Y \cup V_Y$, where $U_X = B_X$, $|V_X| = 1$, and $|U_Y| = |V_Y| = 2$, for each fixed $f(X_i)$, $\in V_X$, including different $f(Y_j)$, $f(Y_l) \in R_Y$ (note $j \ne i$ and $l \ne i$ by Lemma 5) in V_Y only affects $\alpha_3(S_{32})$, $t_j(S_{32})$, and $t_l(S_{32})$. Therefore, we want to choose $f(Y_j)$, $f(Y_l) \in R_Y$ for each fixed $f(X_i) \in R_X$ to minimize $\max\{\alpha_3(S_{32}), t_j(S_{32}), t_l(S_{32})\}$. Set $\{f(Y_j), f(Y_l)\}$ is then a candidate for V^*_Y. Procedure **D**, a variation of procedure **C**, determines S^*_{32}. Since j and l are just indices, without loss of generality, we assume $t_j(S_{32}) \ge t_l(S_{32})$.

Procedure D

1. $S^*_{32} \leftarrow \emptyset \cup \emptyset \cup \emptyset \cup \emptyset$; $U_X \leftarrow B_X$;
2. Create a list L of all choices in R_Y, and for each $f(Y_j) \in L$, define $key(j) = t_j(S_{32})$, where $S_{32} = U_X \cup V_X \cup U_Y \cup V_Y$ with $f(Y_j) \in V_Y$ and $f(X_j) \notin V_X$ (by Lemma 5);
3. Sort L by nondecreasing values of $key(j)$. We call the resulting list L';
4. In L', for each $f(Y_j)$, except the first one, let $f(Y_l)$ be the largest choice among those on the left side of $f(Y_j)$. Now, we have a list of $|R_Y| - 1$ pairs of $f(Y_j)$,$f(Y_l)$ ordered according to $key(j)$ nondecreasingly. We call this list L'';
5. Delete from L'' those pairs with sum $f(Y_j) + f(Y_l)$ no greater than that of their left neighbors in the list. We call the list L''';
6. For each combination of U_X, V_X, U_Y, where $U_X = B_X$, $V_X = \{f(X_i)\ \forall f(X_i) \in R_X$, and $U_Y \subseteq D_Y \cup \{f(Y_i)\}$, do
   Perform a binary search in L''' to locate the pair $f(Y_j)$, $f(Y_l)$ $(j \ne i$ and $l \ne i)$ with the minimum $\max\{\alpha_3(S_{32}), key(j), key(l)\}$, where $S_{32} = U_X \cup V_X \cup U_Y \cup V_Y$ with $V_Y = \{f(Y_j), V(Y_l)\}$;
   $V_Y \leftarrow \{f(Y_j), f(Y_l)\}$;
   $S_{32} \leftarrow U_X \cup V_X \cup U_Y \cup V_Y$;
   If $\alpha(S_{32}) < \alpha(S^*_{32})$ then $S^*_{32} \leftarrow S_{32}$.

The justification for line 4 and line 5 is that if $U_Y = \{f(Y_1), f(Y_2)\}$ and $V_Y = \{f(Y_j), f(Y_l)\}$, then $\alpha_3(S_{32}) = \alpha^*_3 +$

$2n - f(Y_1) - f(Y_2) - f(Y_j) - f(Y_l)$ and it gets smaller when $f(Y_j) + f(Y_l)$ gets larger. Therefore, we always want to choose the largest $f(Y_l)$ for each $f(Y_j)$ as in line 4, and delete some pairs with smaller $f(Y_j) + f(Y_l)$ as in line 5. Obviously, the time complexity for procedure **D** is $O(k \log k)$.

Step 3.2. Determine S_3^* for the case of $|B_X| = 2$.

Assume $S_3^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$ with $V_X^*, V_Y^* \neq \emptyset$. By Lemma 7, $|D_X| \leq 1$ and $|D_Y| \leq 1$, and if there is a disastrous choice in f_X, there is also a disastrous choice in f_Y. So $|D_Y| = 1$, otherwise $|D_X|$ and $|D_Y|$ would both be 0, conflicting our assumption that there is at least one disastrous choice in f_X or f_Y. Assume $D_Y = \{f(Y_1)\}$. We then claim that $|D_X| = 1$. Suppose $|D_X| = 0$, then $f(X_1) \notin B_X$ and by Lemma 7, $f(X_1) \in R_X$. Consider any $f(X_i) \in U_X^*$. Since $f(X_i)$ is potential switch but not a disastrous choice, $f(Y_i)$ must be in V_Y^*. Therefore, $t_i(S_3^*) = m - f(X_i) + n - f(Y_i) \geq f(X_1) + f(Y_1)$. So $\alpha(S_3^*)$ is at least as large as the cost of the original assignment defined by f, and S_3^* cannot be the set of optimal switches we are looking for. Now we have $D_X = \{f(X_1)\}$ and $D_Y = \{f(Y_1)\}$. By Lemma 7, $|U_X^*| = |U_Y^*| = 1$.

Therefore, $|U_X^*| = |V_X^*| = |U_Y^*| = |V_Y^*| = 1$. This is similar to the situation in Step 3.1.1, except that there are two candidates for U_X^* rather than one. Assume $B_X = \{f(X_1), f(X_2)\}$. Let $U_X^* = \{f(X_1)\}$ and call procedure **C**, then let $U_X^* = \{f(X_2)\}$ and call procedure **C**. S_3^* is therefore the better of the two sets of switches found by two calls of procedure **C**. The time complexity is obviously $O(k \log k)$.

5. Conclusions

This paper studies a problem of routing messages on an SIMD parallel architecture whose processing elements (PE) are connected as a toroidal mesh. In our problem the sets of messages processors send are isomorphic, meaning that if some processor i has a message to send which must traverse x_i PEs in the East-West dimension and y_i PEs in the North-South, then all PEs have a message to send with identical routing offsets. We examine variants of the problem having differing assumptions concerning simultaneous use of communication channels, and the ability to buffer a message temporarily en-route. We provide new results not only on the motivating routing problem, but on a new class of scheduling problems as well.

A spectrum of complexities are obtained, from linear in the number of messages (k) per processor to NP-complete. The variation where all ports may be used simultaneously and messages may be buffered en-route is of particular interest. Our solution approach is to view the problem as a scheduling problem, related to a previously studied open shop scheduling problem. We first show why the problem has a polynomial solution, and then present an $O(k \log k)$ implementation. The algorithm lacks elegance; our hope is that future work may provide a more direct solution to the problem.

Acknowledgments

This work was supported in part by NSF grants CCR-9210372 and CCR-9201195.

References

1. E.D. DAHL, 1990. Mapping and Compiled Communication on the Connection Machine System, in *Proceedings of the 5th Distributed Memory Computing Conference.*
2. M.R. GAREY and D.S. JOHNSON, 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco.
3. T. GONZALEZ and S. SAHNI, 1976. Open Shop Scheduling to Minimize Finish Time, *Journal of ACM 23*, 665–679.
4. L. JOHNSON and C. HO, 1986. Spanning Graphs for Optimum Broadcasting and Personalized Communication in Hypercubes, *Yale Computer Science Technical Report TR-500.*
5. E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN, and D.B. SHMOYS, 1990. Sequencing and Scheduling. Algorithms and Complexity, in *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*, S.C. Graves, A.H.G. Rinnooy Kan, and P. Zipkin (eds.), North-Holland.
6. F.T. LEIGHTON, 1992. *An Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufman, San Mateo, CA.
7. G.F. LEV, N. PIPPENGER, and L.G. VALIANT, 1981. A Fast Parallel Algorithm for Routing in Permutation Networks, *IEEE Transactions on Computers C-30(2)*, 93–100.
8. D. REED, L. ADAMS and M. PATRICK, 1987. Stencils and Problem Partitionings: Their Influence on the Performance of Multiple Processor Systems, *IEEE Transactions on Computers C-36(7)*, 845–858.

Appendix

Lemma 1. *If* $|B_X| \geq 3$, *then* $f^* = f$.

Proof. We shall prove that $D_X = \phi$ and $D_Y = \phi$, hence proving $f^* = f$. Suppose that $f(X_1), f(X_2), f(X_3), \ldots$ are the bad choices in f_X, and that $f(X_1)$ is the largest among all. Assume that there is at least one disastrous choice in f_X (alt., f_Y), say, $f(X_i)$ (alt., $f(Y_i)$). Then $f(X_i) + f(Y_i) > \bar{\alpha} \geq \alpha_2^* \geq f(X_1) + f(X_2) + f(X_3)$. So $f(Y_i) > (f(X_1) - f(X_i)) + f(X_2) + f(X_3) \geq f(X_2) + f(X_3) > 2 \times m/2 = m \geq n$, which is impossible. ■

Lemma 2. $|D_Y| \leq 2$.

Proof. Assume $D_Y = \{f(Y_1), f(Y_2), f(Y_3), \ldots\}$. Then at least two of $f(X_1), f(X_2), f(X_3)$ are in the same column in the assignment diagram for f_X, say, $f(X_1)$ and $f(X_2)$. Since both $f(Y_1)$ and $f(Y_2)$ are disastrous choices, we have $f(X_1) + f(Y_1) > \bar{\alpha}$, and $f(X_2) + f(Y_2) > \bar{\alpha}$. So, $f(X_1) + f(Y_1) + f(X_2) + f(Y_2) > 2\bar{\alpha}$. However, we know $f(X_1) + f(X_2) \leq \alpha_2^* \leq \bar{\alpha}$, and $f(Y_1) + f(Y_2) \leq \alpha_3^* \leq \bar{\alpha}$. So, $f(X_1) + f(X_2) + f(Y_1) + f(Y_2) \leq 2\bar{\alpha}$. This is a contradiction. ■

Lemma 3. *In* $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$, $|U_X^*| \geq |V_X^*|$ *and* $|U_Y^*| \geq |V_Y^*|$.

Proof. We only prove $|U_X^*| \geq |V_X^*|$, since the proof of $|U_Y^*| \geq |V_Y^*|$ is totally symmetric and hence can be omitted. For any given $S = U_X \cup V_X \cup U_Y \cup V_Y$, where $|U_X| < |V_X|$, let $S' = U_X \cup V_X' \cup U_Y \cup V_Y$, where $V_X' \subset V_X$ and $|V_X'| = |U_X|$. Let $\alpha_2^* = \max\{\alpha_2^+, \alpha_2^-\}$, where $\alpha_2^+ = \sum_{f(X_i)=x_i} x_i$, and

$\alpha_2^- = \sum_{f(X_i)=m-x_i} (m - x_i)$.

$$\alpha_2(S) = \max\left\{ \alpha_2^+ - \sum_{U_X} f(X_i) + \sum_{V_X} (m - f(X_i)), \right.$$
$$\left. \alpha_2^- - \sum_{V_X} f(X_i) + \sum_{U_X} (m - f(X_i)) \right\}$$
$$= \alpha_2^+ - \sum_{U_X} f(X_i) + \sum_{V_X} (m - f(X_i))$$
(because $\alpha_2^+ - \alpha_2^- > -m \geq m(|U_X| - |V_X|)$).

$$\alpha_2(S') = \max\left\{ \alpha_2^+ - \sum_{U_X} f(X_i) + \sum_{V_X'} (m - f(X_i)), \right.$$
$$\left. \alpha_2^- - \sum_{V_X'} f(X_i) + \sum_{U_X} (m - f(X_i)) \right\}$$
$$= \alpha_2^* - \sum_{U_X} f(X_i) + \sum_{V_X'} (m - f(X_i))$$
$$\leq \alpha_2^+ - \sum_{U_X} f(X_i) + \sum_{V_X} (m - f(X_i))$$
$$\left(\text{because } \alpha_2^- - \alpha_2^+ \leq \sum_{V_X - V_X'} (m - f(X_i)) \text{ if } \alpha_2^- > \alpha_2^+ \right)$$
$$= \alpha_2(S).$$

Since all choices in V_X' and V_X become bad choices after they are switched and $V_X' \subset V_X$, then $\alpha_1(S') \leq \alpha_1(S)$. For α_3, it remains the same in S and S'. So $\alpha_3(S') = \alpha_3(S)$. We then have $\alpha(S') = \max\{\alpha_1(S'), \alpha_2(S'), \alpha_3(S')\} \leq \max\{\alpha_1(S), \alpha_2(S), \alpha_3(S)\} = \alpha(S)$.

We conclude that for any $S = U_X \cup V_X \cup U_Y \cup V_Y$ with $|U_X| < |V_X|$, there is always $S' = U_X \cup V_X' \cup U_Y \cup V_Y$ with $|U_X| \geq |V_X'|$ such that $\alpha(S') \leq \alpha(S)$. Therefore, there exists $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$ for the optimal assignment such that $|U_X^*| \geq |V_X^*|$. ■

Lemma 4. *In $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$, all members of U_X^* and U_Y^* are potential switches.*

Proof. We only prove the lemma for U_X^*. We first need the following result: For any $S = U_X \cup V_X \cup U_Y \cup V_Y$ with $|U_X| \geq |V_X|$, let $S' = U_X' \cup V_X' \cup U_Y \cup V_Y$, where $U_X' \subset U_X$, $V_X' \subset V_X$ and $|V_X'| = \max\{0, |V_X| - |U_X| + |U_X'|\}$. Then $\alpha_2(S') < \alpha_2(S)$. To prove this, we consider two cases.

Case 1. If $|U_X| = |V_X|$, then $|U_X'| = |V_X'|$, and

$$\alpha_2(S) = \max\left\{ \alpha_2^+ - \sum_{U_X} f(X_i) + \sum_{V_X} (m - f(X_i)), \right.$$
$$\left. \alpha_2^- - \sum_{V_X} f(X_i) + \sum_{U_X} (m - f(X_i)) \right\}$$
$$= \alpha_2^* - \sum_{U_X} f(X_i) + \sum_{V_X} (m - f(X_i)),$$

$$\alpha_2(S') = \max\left\{ \alpha_2^+ - \sum_{U_X'} f(X_i) + \sum_{V_X'} (m - f(X_i)), \right.$$
$$\left. \alpha_2^- - \sum_{V_X'} f(X_i) + \sum_{U_X'} (m - f(X_i)) \right\}$$
$$= \alpha_2^* - \sum_{U_X'} f(X_i) + \sum_{V_X'} (m - f(X_i))$$
$$\leq \alpha_2^* - \sum_{U_X} f(X_i) + \sum_{V_X} (m - f(X_i))$$
$$\left(\text{because } \sum_{U_X - U_X'} f(X_i) \leq \sum_{V_X - V_X'} (m - f(X_i)) \right)$$
$$= \alpha_2(S).$$

Therefore, $\alpha_2(S') \leq \alpha_2(S)$.

Case 2. If $|U_X| > |V_X|$, then $|U_X'| > |V_X'|$ and $|U_X| - |U_X'| \geq |V_X| - |V_X'|$, and

$$\alpha_2(S) = \max\left\{ \alpha_2^+ - \sum_{U_X} f(X_i) + \sum_{V_X} (m - f(X_i)), \right.$$
$$\left. \alpha_2^- - \sum_{V_X} f(X_i) + \sum_{U_X} (m - f(X_i)) \right\}$$
$$= \alpha_2^- - \sum_{V_X} f(X_i) + \sum_{U_X} (m - f(X_i))$$
(because $\alpha_2^+ - \alpha_2^- < m \leq m(|U_X| - |V_X|)$),

$$\alpha_2(S') = \max\left\{ \alpha_2^+ - \sum_{U_X'} f(X_i) + \sum_{V_X'} (m - f(X_i)), \right.$$
$$\left. \alpha_2^- - \sum_{V_X'} f(X_i) + \sum_{U_X'} (m - f(X_i)) \right\}$$
$$= \alpha_2^- - \sum_{V_X'} f(X_i) + \sum_{U_X'} (m - f(X_i))$$
(because $\alpha_2^+ - \alpha_2^- < m \leq m(|U_X'| - |V_X'|)$)
$$\leq \alpha_2^- - \sum_{V_X} f(X_i) + \sum_{U_X} (m - f(X_i))$$
$$\left(\text{because } \sum_{V_X - V_X'} f(X_i) \leq \sum_{U_X - U_X'} (m - f(X_i)) \right)$$
$$= \alpha_2(S).$$

Therefore, $\alpha_2(S') \leq \alpha_2(S)$.

Now we are ready to prove Lemma 4. First we show that all members of U_X^* are bad choices. Suppose not, let $U_X' \subset U_X^*$ be the set of all bad choices in U_X^*, and $V_X' \subset V_X^*$ be such that $|V_X'| = \max\{0, |V_X^*| - |U_X^*| + |U_X'|\}$. Let $S' = U_X' \cup V_X' \cup U_Y^* \cup V_Y^*$. Using the result just proved, we have $\alpha_2(S') \leq \alpha_2(S^*)$. Since all choices in $U_X^* - U_X'$ and $V_X^* - V_X'$ become bad choices after they are switched, $\alpha_1(S') \leq \alpha_1(S^*)$. Further-

more, $\alpha_3(S') = \alpha_3(S^*)$. We then have $\alpha(S') = \max\{\alpha_1(S'), \alpha_2(S'), \alpha_3(S')\} \le \max\{\alpha_1(S^*), \alpha_2(S^*), \alpha_3(S^*)\} = \alpha(S^*)$. Since S^* is the set of switches that determines the optimal assignment f^*, S' must also be one that determines f^*. We then conclude that there exists $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$, where all members of U_X^* are bad choices.

Next we show that all members of U_X^* are potential switches. Suppose not, let $U_X' \subset U_X^*$ be the set of all potential switches in U_X^*, and $V_X' \subset V_X^*$ be such that $|V_X'| = \max\{0, |V_X^*| - |U_X^*| + |U_X'|\}$. Let $S' = U_X' \cup V_X' \cup U_Y^* \cup V_Y^*$. Obviously, $\alpha_2(S') \le \alpha_2(S^*)$. Consider any choice $f(X_i) \in U_X^* - U_X'$. Since $f(X_i)$ is a bad choice but not a potential switch, by the definition of potential switches we must have $f(X_i) + f(Y_i) \le \bar{\alpha}$ and either $f(Y_i) \notin V_Y^*$ or $f(X_i) + n - f(Y_i) \le \alpha_3(S^*)$. Consider $t_i(S')$, the term contributed to $\alpha_1(S')$ by tasks X_i and Y_i. If $f(Y_i) \notin U_Y^* \cup V_Y^*$, then $t_i(S') = f(X_i) + f(Y_i) \le \bar{\alpha}$. If $f(Y_i) \in U_Y^*$, then $f(Y_i)$ must be a bad choice and $t_i(S') = f(X_i) + n - f(Y_i) < f(X_i) + f(Y_i) \le \bar{\alpha}$. If $f(Y_i) \in V_Y^*$, then $t_i(S') = f(X_i) + n - f(Y_i) \le \alpha_3(S^*)$. So $t_i(S') \le \max\{\bar{\alpha}, \alpha_3(S^*)\}$. Furthermore, all choices in $V_X^* - V_X'$ become bad choices after they are switched. So $\alpha_1(S') \le \max\{\alpha_1(S^*), \bar{\alpha}, \alpha_3(S^*)\}$. Since $\alpha_3(S') = \alpha_3(S^*)$, we then have $\alpha(S') = \max\{\alpha_1(S'), \alpha_2(S'), \alpha_3(S')\} \le \max\{\alpha_1(S^*), \alpha_2(S^*), \alpha_3(S^*), \bar{\alpha}\} = \max\{\alpha_1(S^*), \alpha_2(S^*), \alpha_3(S^*)\} = \alpha(S^*)$. S' must also be the set of switches that determines the optimal assignment f^*. We then conclude that there exists $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$, where all members of U_X^* are potential switches. ■

Lemma 5. *In $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$, there are no tasks X_i and Y_i such that $f(X_i) \in V_X^*$ and $f(Y_i) \in V_Y^*$.*

Proof. Let j be any index for which either $f(X_j)$ or $f(Y_j)$ is a disastrous choice. Assume $f(X_i) \in V_X^*$ and $f(Y_i) \in V_Y^*$ for some i. We observe that $m - f(X_i) \ge m/2$ and that $m - f(X_i)$ is at least as large as any choice in f_X. So $m - f(X_i) \ge f(X_j)$. Similarly, $n - f(Y_i) \ge f(Y_j)$. Therefore, $m - f(X_i) + n - f(Y_i) \ge f(X_j) + f(Y_j)$ for any disastrous choice with index j. So the assignment defined by $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$ has cost at least as large as the original assignment defined by f. We certainly need not consider such assignment. ■

Lemma 6. *Suppose $|B_X| = 1$. Then in $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$, $|V_X^*| \le |U_X^*| \le 1$ and $|V_Y^*| \le |U_Y^*| \le 2$. Furthermore, if $D_Y = \{f(Y_1), f(Y_2)\}$, then $\bar{\alpha} < f(X_1) + f(X_2)$ and one of $f(X_1)$ and $f(X_2)$ is the only bad and disastrous choice in f_X.*

Proof. Let us first prove the second part of the lemma. Assume $D_Y = \{f(Y_1), f(Y_2)\}$. Suppose $\bar{\alpha} \ge f(X_1) + f(X_2)$, then $f(X_1) + f(Y_1) > \bar{\alpha} \ge f(X_1) + f(X_2)$. So $f(Y_1) > f(X_2)$. On the other hand, $f(X_2) + f(Y_2) > \bar{\alpha} \ge \alpha_3^* \ge f(Y_1) + f(Y_2)$. So $f(X_2) > f(Y_1)$. A contradiction! So $\bar{\alpha} < f(X_1) + f(X_2)$, and $f(X_1)$ and $f(X_2)$ must be in the different columns, and the one in the left column, say, $f(X_1)$, has to be the only bad choice in f_X. This also implies that $D_X = \{f(X_1)\}$.

Let us prove the first part of the lemma. By Lemma 3 and Lemma 4, $|V_X^*| \le |U_X^*| \le |B_X| = 1$. By Lemma 3, $|V_Y^*| \le |U_Y^*|$. Suppose that $|U_Y^*| \ge 3$. By Lemma 4, U_Y^* contains only potential switches. So $3 \le |U_Y^*| \le |D_Y| + |V_X^*| \le |D_Y| + 1$. We have $|D_Y| \ge 2$. However, by Lemma 2, $|D_Y| \le 2$. So $|D_Y| = 2$. Assume $D_Y = \{f(Y_1), f(Y_2)\}$. By the second part of this lemma which we just proved, assume $D_X = \{f(X_1)\}$. Since $3 \le |U_Y^*| \le |D_Y| + 1 = 3$, $|U_Y^*| = 3$. Assume $U_Y^* = D_Y \cup \{f(Y_i)\}$, where $f(Y_i) \in B_Y$ ($i \ne 1, 2$), $f(X_i) + f(Y_i) \le \bar{\alpha}$, and $f(X_i) \in V_X^*$. We can show that $t_i(S^*) > \max\{f(X_1) + f(Y_1), f(X_2) + f(Y_2)\}$. To do so, we have $f(X_2) + f(Y_2) > \bar{\alpha} \ge \alpha_3^* \ge f(Y_1) + f(Y_2) + f(Y_i)$, therefore $f(X_2) > f(Y_1) + f(Y_i)$. Since $f(X_1) \le m - f(X_2) < m - f(Y_1) - f(Y_i) < m - f(Y_1) - f(Y_i) + n - f(X_i)$, then $t_i(S^*) = m - f(X_i) + n - f(Y_i) > f(X_1) + f(Y_1)$. Since $f(X_2) + f(X_i) \le \alpha_2^* \le \bar{\alpha} < f(X_1) + f(X_1) < f(X_1) + f(X_2) - f(Y_i) \le m - f(Y_i) < m - f(Y_i) + n - f(Y_2)$, then $t_i(S^*) = m - f(X_i) + n - f(Y_i) > f(X_2) + f(Y_2)$. This implies that the assignment defined by $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$ has cost greater than the original assignment defined by f. So S^* cannot be the set of optimal switches. ■

Lemma 7. *Suppose $|B_X| = 2$. Then $|D_X| \le 1$ and $|D_Y| \le 1$. Furthermore, if $D_X = \{f(X_1)\}$, then $D_Y = \{f(Y_1)\}$ and in $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$, $|U_X^*| = |U_Y^*| = 1$; if $D_Y = \{f(Y_1)\}$ and $f(X_1) \notin B_X$, then $f(X_1) \in R_X$.*

Proof. Suppose $B_X = \{f(X_i), f(X_j)\}$. First, we notice that if $f(X_l)$ ($l = i$ or j) is a disastrous choice, $f(Y_l)$ is also a disastrous choice, because $f(X_l) + f(Y_l) > \bar{\alpha} \ge \alpha_2^* \ge f(X_i) + f(X_j)$, and therefore $f(Y_l) > m/2 \ge n/2$.

Assume that there are at least two disastrous choices in f_X. They must be $f(X_i)$ and $f(X_j)$. Since both $f(Y_i)$ and $f(Y_j)$ are disastrous choices, they are in the same column in assignment diagram of f_Y. Then $f(X_i) + f(Y_i) > \bar{\alpha} \ge \alpha_3^* \ge f(Y_i) + f(Y_j)$. So $f(X_i) > f(Y_j)$. On the other hand, $f(X_j) + f(Y_j) > \bar{\alpha} \ge \alpha_2^* \ge f(X_i) + f(X_j)$. So $f(Y_j) > f(X_i)$. A contradiction!

Assume that there are at least two disastrous choices in f_Y, say, $f(Y_1)$ and $f(Y_2)$. Then $f(X_1) + f(Y_1) > \bar{\alpha} \ge \alpha_3^* \ge f(Y_1) + f(Y_2)$. So $f(X_1) > f(Y_2)$. On the other hand, $f(X_2) + f(Y_2) > \bar{\alpha} \ge \alpha_2^* \ge f(X_i) + f(X_j) \ge f(X_1) + f(X_2)$. So $f(Y_2) > f(X_1)$. A contradiction!

If $D_X = \{f(X_1)\}$, then $D_Y = \{f(Y_1)\}$. Now consider $S^* = U_X^* \cup V_X^* \cup U_Y^* \cup V_Y^*$. We observe that S^* does not contain both $f(X_1)$ and $f(Y_1)$. Because assuming $B_X = \{f(X_1), f(X_2)\}$, $m - f(X_1) + n - f(Y_1) < f(X_1) + n - f(Y_1) < f(X_1) + n/2 \le f(X_1) + m/2 < f(X_1) + f(X_2) \le \alpha_2^* \le \bar{\alpha}$, which suggests that switching just $f(Y_1)$ is already good enough, why bother to switch both $f(X_1)$ and $f(Y_1)$? We next show that $|U_X^*| = 1$. Suppose not, then $U_X^* = B_X = \{f(X_1), f(X_2)\}$ and $f(Y_1) \notin U_Y^*$. Since U_X^* contains only potential switches and $f(X_2)$ is not a disastrous choice, we must have $f(Y_2) \in V_Y^*$ and $f(X_2) + n - f(Y_2) > \alpha_3(S_3^*)$. Furthermore, since the only disastrous choice $f(Y_1)$ in D_Y is not a member of U_Y^*, there must be $f(Y_3) \in U_Y^*$. Because $\alpha_3(S^*) \ge \alpha_3^* + n - f(Y_3) - f(Y_2)$, then $f(Y_1) + f(Y_3) \le \alpha_3^* \le \alpha_3(S^*) - n + f(Y_2) + f(Y_3) < f(X_2) + f(Y_3)$. So $f(Y_1) < f(X_2)$. On the other hand, $f(X_1) + f(Y_1) > \bar{\alpha} \ge f(X_1) + f(X_2)$. So $f(Y_1) > f(X_2)$. A contradiction! Similarly, we can prove $|U_Y^*| = 1$.

If $D_Y = \{f(Y_1)\}$ and $f(X_1) \notin B_X$, then $f(X_1) \in R_X$. Otherwise, assuming that $B_X = \{f(X_i), f(X_j)\}$ ($i \ne 1$ and $j \ne 1$), $f(X_1) + f(Y_1) > \bar{\alpha} \ge \alpha_2^* \ge f(X_i) + f(X_j) + f(X_1)$. So $f(Y_1) > f(X_i) + f(X_j) > m \ge n$, which is impossible. ■