Talking Fast: Finding the Makespan of a Communications Network

Matthew Prochazka
Wendy Schaub
Gary Amende
Beloit College
Beloit, WI 53511–5595

Advisor: Philip D. Straffin

Summary

We were asked to find an optimal schedule and makespan for three different situations, as well as to provide a generalized algorithm for the optimal schedule and makespan.

To check all the possible schedules is very inefficient and time-consuming and, on a large network, could take several days or weeks. Instead, we implemented a greedy algorithm, based on the expectation that optimizing locally will result in global optimization.

The makespan will be at least as long as it takes for any one computer in its network to finish its data transfers. So the computer that takes the longest to finish its data transfers should start transferring first. By choosing the best possible transfer at each step in the algorithm, the schedule was progressively built up into a prediction for an optimal schedule and makespan.

We found optimal schedules and makespans (3, 23.0, and 29.6) for situations A, B, and C. We also proved the correctness of these makespans found by our algorithm. Our method can be applied to a general network; although our computer program does not guaranteed optimality, a near-optimal schedule and makespan are produced.

For situations A and B, we recommend implementing the optimal schedules. For situation C, we recommend switching some computers capable of transferring more than one file at a time to locations that require multiple file transfers. In situation C, if each computer transfers only one file at a time, the makespan remains the same; but if the transfer times do not change when computers are switched, the makespan can be improved by 35%.

Assumptions

- Each network has computers, represented by vertices ($V_1, V_2, \ldots, V_m$) and transfers between them, represented by edges ($e_1, e_2, \ldots, e_n$). Each computer has a designated capacity, $C(V_i)$, to transfer files simultaneously, and each transfer has a designated time necessary to complete, $T(e_j)$.
- The computers do not break down and are always functioning.
- All computers in the network must receive the information on a daily basis.
- Equal priority is placed on all files.
- All transfers are made directly, not via a third party. Thus, all computers have the necessary information to make the transfer. This assumption avoids the problem of waiting for information from another computer before a transfer can be made (which would obviously increase the makespan).
- The process of transferring files does not have to wait for the implementation of this model. This assumption avoids any delays that depend on the running time of the computer model.
- Given a schedule, the company knows how to implement it.

Analysis of the Problem

The minimum time required by each computer can be calculated using its capacity and the number of connecting transfers. For example, computer V_{20} in **Figure 2** of the problem statement must transfer three files, each of which requires seven time units to complete. Since the computer can transfer only one file at a time, the file transfers must be done sequentially, producing a minimum total time of 21 time units. In this way, a minimum time can be found for each computer in the network.

The makespan for this network will be at least the largest of these minima, since each computer must complete all of its transfers. A starting point would be to initiate transfers that require the maximum time to complete. Once those computers are transferring data, the constraints of the network indicate which others can begin transferring at the same time. When a transfer is complete, two computers become available to work on other transfers. At this time, a re-evaluation of the network is needed to determine if another transfer can be initiated. By re-evaluating the network every time a transfer is completed, the idle time is reduced to a minimum. In this way, a minimum overall transfer time can be achieved.

Model Design

We decided to solve this problem using a greedy algorithm. This is an optimization algorithm that applies a simple-minded strategy of progressively building up a solution, one element at a time, by choosing the best possible element at each iteration. Greedy algorithms are based on the hope that optimizing locally (at each step, we do as well as we can) will result also in global optimization (at the end, we have an optimal solution overall). In our case, our elements were computers and we were trying to choose the best possible transfers to initiate at each iteration to schedule the makespan.

We first had to arrange the computers in a priority list. Doing so allowed us to choose the two best possible computers to transfer data between at each iteration. We used a modified bin-packing algorithm, which calculates the minimum time needed for a computer to finish (complete its transfers).

Modified Bin-Packing Algorithm: For each computer V_i, let the number of bins be $c = C(V_i)$. Each V_i must perform a subset of the set of all transfers in the network.

1. Suppose that V_i must perform k transfers. Rename these transfers $e_{(1)}, \ldots, e_{(k)}$ so that $T(e_{(1)}) \leq T(e_{(2)}) \leq \cdots \leq T(e_{(k)})$.
2. If $k \leq C(V_i)$, place one transfer in each bin. Thus, $T(e_{(k)})$ is the minimum time for that computer to finish.
3. If $k > C(V_i)$, place $T(e_{(k)})$ in the first bin, $T(e_{(k-1)})$ in the second, and so on, until all the bins contain one transfer. Place the next longest transfer, $T(e_{(k-c-1)})$, in the bin containing the lowest transfer time, $T(e_{(k-c)})$, and sum their two times. Place the next longest transfer, $T(e_{(k-c-2)})$ in the bin containing the lowest summed transfer time, and sum their times. Continue in this way until all transfers are placed in bins. The bin with the greatest summed transfer time is the minimum time for that computer to finish.

Using this algorithm, the computer that requires the most time to finish gives a lower bound for the entire network transfer time.

By placing the minimum transfer times needed by each computer in ascending order, we obtain a priority list; and the greedy algorithm can be used to find a minimal schedule. To initiate transfers, the two computers with the longest minimum transfer times are connected if a connection is possible. If no connection exists between these two computers, a connection to the next computer on the priority list is attempted. This continues until a successful connection is made.

At this stage, there are two possibilities for further connections: Either the capacity of the first computer on the priority list is equal to 1, or it is greater than 1. In the case where the capacity equals 1, the computer is

no longer available to make other transfers. Therefore, the first available computer on the priority list is connected to the next computer on the list to which a connection is possible. This continues until all possible connections are made.

If the capacity is greater than 1 for the computer requiring the greatest transfer time, the computer is available for other transfers. A connection is then attempted between this computer and the next available computer on the priority list, until either the number of connections made by the first computer equals its capacity or no other connections are possible. Therefore, all possible transfers involving this computer are in progress. This procedure is repeated for each computer, working in descending order of the priority list until all possible connections are made and a minimal schedule achieved.

Computer Implementation

We developed a Pascal computer program to enable the company to analyze their computer network and obtain a minimal schedule. The program produces the minimum time for each computer to finish, calculates a schedule of times to start each of the transfers, and reports the total elapsed time of the minimal schedule. This program can be analyzed using big-oh notation [Sedgwick 1988]:

Definition. *A function $g(N)$ is said to be $O(f(N))$ if there exist constants c_o and N_o such that $g(N) < c_o f(N)$ for all $N > N_o$.*

For a problem of a certain size, the O-notation is a useful way to state upper bounds on running time that are independent of both inputs and implementation details. In our case, the upper bounds are independent of computer capacities and transfer times between computers. We found that our program has a running time of $O(m^3)$ for m computers. Closer examination shows that in the worst-case scenario, the constant c_o is close to $1/4$; in the average case, however, this constant is much smaller.

Situation A

The corporation has 28 computers, each with capacity 1, and 27 transfers, each with transfer time of 1 unit. With our algorithm, we found the makespan to be 3 units. The makespan cannot be any lower than this, because any computer on the interior of the network requires at least 3 units to complete its transfers. There is more than one way to optimally schedule the transfers, such as the schedule shown in **Table 1**.

Table 1.
Schedule for situation A.

At time	Start transfers
0.0	$e_2, e_5, e_{10}, e_{13}, e_{16}, e_{21}, e_{25}$
1.0	$e_1, e_3, e_7, e_9, e_{11}, e_{14}, e_{17}, e_{20}, e_{23}, e_{27}$
2.0	$e_4, e_6, e_8, e_{12}, e_{15}, e_{18}, e_{19}, e_{22}, e_{24}, e_{26}$
	Total time elapsed: 3.0 units

Situation B

The network is the same as in situation A, but the transfer times are different. The maximum time required by any of the computers is 21 units, which is the minimum time required by computer V_{20} to finish. Thus, a lower bound of the makespan for this network is 21 units. Our program, however, predicts a makespan of 23 units, as seen in **Table 2**.

Table 2.
Schedule for situation B.

At time	Start transfers
0.0	$e_4, e_6, e_{10}, e_{13}, e_{16}, e_{20}, e_{24}, e_{26}$
5.0	e_{23}
7.0	e_3, e_{22}
8.0	$e_2, e_9, e_{12}, e_{15}, e_{18}$
9.0	e_7, e_{19}, e_{27}
11.2	e_{21}
12.1	e_1
12.2	e_5
12.4	e_{11}
12.5	e_{17}
13.0	e_8
16.0	e_{14}, e_{25}
	Total time elapsed: 23.0 units

The discrepancy can be explained by observing the architecture of the network. Computer V_{20} requires a minimum of 21 units to finish (e_{19}, e_{20}, e_{25}), because each transfer requires 7 units and the computer can transfer only one file at a time. This is the maximum time required by any of the computers in the network, so it is a lower bound.

To achieve this lower bound, the first transfer should include V_{20}. Looking at all the possible transfers involved with V_{20}, one could start with e_{20}. When e_{20} finishes, e_{19} or e_{25} should be initiated immediately, and thus e_{13} and e_{26} cannot both be transferring at this time. If e_{13} is not transferring, e_{19} would be initiated, which takes 7 units to complete. Now, e_{13} cannot

begin transferring until these 7 units have elapsed. So after 14 units (7 from e_{20} and 7 from e_{19}), e_{13} can begin. Transfer e_{13} takes 9 units to complete; so by the time that it is done, 23 units will have elapsed. Thus, 23 units is a lower bound based on this network's limitations. Similar situations arise when e_{19} and e_{25} are initiated first. If the initial transfer does not include a transfer connected to V_{20}, then V_{20} sits idle, so the overall time the schedule takes is greater than 23 time units. So the makespan for this network is 23 units, which agrees with our program's prediction.

Situation C

The network of 28 computers is expanded to include 42 transfers with varied transfer times and capacities. Computer V_{24}, which has five connecting transfers and the capacity to transfer only one file at a time, requires the maximum time to finish of all the computers; so a lower bound for the network to finish is the sum of V_{24}'s transfers, 29.6 units. The schedule produced by our program, shown in **Table 3**, takes 29.6 units, so this schedule runs at the makespan.

Table 3.
Schedule for situation C.

At time	Start transfers
0.0	e_4, e_6, e_{10}, e_{11}, e_{12}, e_{13}, e_{15}, e_{17}, e_{18}, e_{20}, e_{24}, e_{26}, e_{35}, e_{41}
2.1	e_{32}
3.0	e_9, e_{14}
4.4	e_{28}
5.0	e_{23}, e_{36}
6.1	e_{31}
6.2	e_{16}
7.0	e_3, e_{19}, e_{33}, e_{38}
8.0	e_1, e_{40}
9.0	e_8
10.2	e_{30}
11.0	e_{37}, e_{39}
14.0	e_{21}, e_{27}, e_{42}
14.2	e_{29}
15.1	e_5
17.3	e_2
20.1	e_7, e_{25}
23.0	e_{34}
25.4	e_{22}
Total time elapsed: 29.6 units	

Strengths and Weaknesses

The greatest strength of our model is that the schedules produced run at the makespan for all three situations. This algorithm works well on all networks in which the computer that requires the maximum transfer time is capable of transferring only one file at a time. When this computer has the capacity of simultaneously transferring more than one file, the modified bin-packing algorithm used to find a lower bound starts to break down, as seen in the **Example** below. Even if we don't find the makespan, we get a schedule that runs in near-makespan time.

Our model is completely adaptable to other networks; and expansion or reduction of transfers, computers, or capacities is easily accommodated.

One weakness of our model is that all files are given equal priority (i.e., there is no capacity for a "rush transfer"). Each file transfer gets scheduled, but the transfers may not occur in a desired order.

This weakness goes hand in hand with the assumption that all computers contain the necessary information to complete their transfers. This may not be the case in all companies; e.g., computer B cannot transfer information to computer C until it receives its files from computer A. Our model does not take this type of network into account.

Another weakness of our model is the $O(m^3)$ running time.

Example. Say that the computer requiring the maximum transfer time also can simultaneously transfer three files. The eight connecting transfers and their times are shown in **Table 4**.

Table 4.
The eight connecting transfers and their times.

Edge	Transfer time
e_1	9.0
e_2	8.6
e_3	7.2
e_4	4.7
e_5	4.3
e_6	4.1
e_7	3.7
e_8	2.0

Using our modified bin-packing algorithm, e_1, e_2, and e_3 would all be placed in separate bins. Then

- e_4 would be placed in the same bin as e_3, summing to 11.9;
- e_5 would go in the bin with e_2, summing to 12.9;
- e_6 would go in the same bin as e_1, summing to 13.1;
- e_7 would go in the bin with e_3 and e_4, summing to 15.6; and

- e_8 would go in the bin with e_2 and e_5, summing to 14.9.

The picture would look something like **Table 5**.

Table 5.
Result after the modified bin-packing algorithm.

Bin 1	Bin 2	Bin 3
9.0	8.6	7.2
4.1	4.3	4.7
	2.0	3.7
13.1	14.9	15.6

By exchanging the 4.7 in bin 3 with the 4.1 in bin 1, the maximum time required by this computer would be 15.0 instead of 15.6.

Table 6.
Result after exchange.

Bin 1	Bin 2	Bin 3
9.0	8.6	7.2
4.7	4.3	4.1
	2.0	3.7
13.7	14.9	15.0

Sensitivity Testing

To test the algorithm, we considered changes in the situations:

- If the number of processes per computer is changed, the makespan would change accordingly. In situation C, when we reduced the capacity of the all the computers to 1, the makespan stayed exactly the same.
- If only the number of computers is changed, the algorithm would produce a schedule that runs in makespan or near-makespan time.
- If the number of transfers is increased or decreased, the algorithm again would adjust accordingly. In one example, however, an increase in the number of transfers did not affect the makespan: In situation B, when e_{28}, e_{29}, e_{30}, e_{31}, e_{32}, e_{36}, e_{38}, e_{39}, and e_{41} (as defined in situation C) were added to the network, the makespan stayed the same.
- Even if all of the computers in the network transfer at the same time, the makespan may be reduced if hardware is used efficiently. For example, we were able to reduce the makespan in situation C (see **Table 3**) by

making a few computer exchanges. By simply exchanging V_2 and V_{24}, V_9 and V_{20}, V_1 and V_3, and V_8 and V_{22}, the makespan can be reduced from 29.6 to 19.3 units. This improves the makespan by 35%.

Recommendations

We recommend that the company run the match program before any actual transfers take place. With the output of the program, they should set up the network so that the transfers can occur exactly as suggested. This eliminates the delay in time required by the implementation of the matching program.

We also recommend that the company make the most efficient use of its computers by putting the computers capable of the most transfers in the locations that require the most transfers. This recommendation is made only under the assumption that all these computers run at the same speed.

References

Foulds, L.R. 1992. *Graph Theory Applications.* New York: Springer-Verlag.

Helman, P., R. Veroff, and F. Carrano. 1991. *Intermediate Problem Solving and Data Structures: Walls and Mirrors.* New York: Benjamin/Cummings.

Sedgwick, R. 1988. *Algorithms.* 2nd ed. New York: Addison-Wesley.

Wilson, R.J. 1985. *Introduction to Graph Theory.* 3rd ed. Essex, England: Longman Group Ltd.