

A Time–Path Scheduling Problem (TPSP) for Aggregating Large Data Files from Distributed Databases using an Optical Burst-Switched Network

Amitabha Banerjee, Narendra Singhal, Jing Zhang, Dipak Ghosal, Chen-Nee Chuah, and Biswanath Mukherjee
University of California, Davis, CA 95616, USA

Email: abanerjee@ucdavis.edu, {singhaln, zhangj, ghosal}@cs.ucdavis.edu, chuah@ece.ucdavis.edu, mukherje@cs.ucdavis.edu

Abstract— We consider the problem of aggregating large data files from distributed databases and address the corresponding challenges involved from a network architecture perspective. We model this problem as one of identifying a time-path schedule (TPS) in a graph representation of the network. We prove that the TPS problem (TPSP) is NP-complete. We then propose a Mixed Integer Linear Programming (MILP)-based approach and three heuristics – Longest-File-First (LFF), Disjoint-Paths (DP), and Most-Distant-File-First (MDFF) – to solve TPSP.

I. INTRODUCTION

A distributed database is a collection of databases located at different geographic locations and connected through a network. Distributed databases have the advantage of being much more scalable than centralized databases. They are attractive in today’s computing environment because many businesses have data warehouses located in multiple locations around the world. This has inspired a lot of research activity on various aspects of distributed databases [2]. However, there has not been much focus on developing efficient network algorithms and protocols to support distributed databases in previous work.

Our work is motivated by the requirements of the “Genomes To Life” (GTL) application, e.g., please see the GTL project of the United States Department of Energy (DoE) [1]. The data related to different aspects of a biological system is analyzed, processed, and stored at different data warehouses. Some of these data warehouses are several hundred Gigabytes in size. From time to time, a supercomputer at a site of scientific experiments may need to aggregate data from some of these data warehouses before its computations. Data aggregation is done at run time, and hence the network is the bottleneck in the computation. Even a single second of idle time, during which data is being aggregated, represents the loss of several teraflops of computing power [1]. Therefore, minimizing the delay in data aggregation is the key to improving the overall system throughput.

We first seek to address the underlying network technology. We believe that Optical Burst Switching (OBS) [3] is a technology which is well suited for this application. We expect the file sizes required from each data warehouse to be considerably large. This makes it very convenient to assemble a data burst, unlike other OBS applications which require efficient burst-

assembly algorithms. Traditional lightpath routing is not very efficient in this case because, with current technology, it requires significant time for the lightpaths to be established before they can be used. Similarly, optical packet switching is not efficient because there exist huge files of data and not packets of data. OBS helps in achieving easy setup of paths for the bursts, without much overhead. After a file has been transferred along a link, an OBS switch may be easily reconfigured to use the link for a different file transfer.

Our work assumes an underlying OBS backbone network connecting the data warehouses. However, we believe that our work is generic enough to serve as a guideline for applications in other underlying network technologies as well.

II. PROBLEM FORMULATION

We consider an OBS mesh network topology, which may be represented as a graph $G(V, E)$, as shown in Figure 1. Vertices V represent OBS nodes, and the edges E represent optical links connecting the OBS nodes. We assume that all the optical links have the same capacity C (say OC-192), and support a single wavelength.

Each data warehouse is connected to a particular OBS node through a dedicated link of capacity C . There may be multiple data warehouses connected to one OBS node. Similarly, the supercomputer is connected to a particular OBS node through dedicated links. All the above links being dedicated are not represented in the graph.

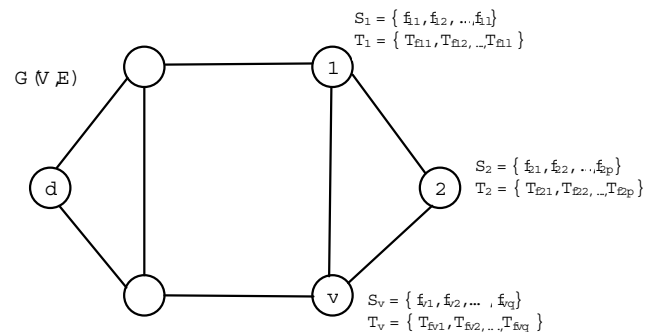


Fig. 1. Graph representation of TPSP.

At a certain step in the computation, the supercomputer may require data aggregated from multiple data warehouses before it resumes computation. We model this process as the transfer of *files* which require to be sent from the source OBS node (to which the corresponding data warehouses are connected) to the destination supercomputer. A query is first issued by the supercomputer to all data warehouses to determine the file size required from each warehouse. The file size provides information on its expected transmission delay. The time that it takes to transfer a file along a route is:

$$T_f = s_f/C + P_d + O_c \quad (1)$$

where T_f and s_f denote the total transfer time and the size for file f , respectively, and C denotes the capacity of each link. P_d is the maximum value of end-to-end propagation delay, and O_c is the control packet overhead.

At each OBS node v , there exist a set of files $S_v = \{f_{v1}, f_{v2}, \dots, f_{vl}\}$ whose T_f is pre-computed and denoted by the set $T_v = \{T_{f_{v1}}, T_{f_{v2}}, \dots, T_{f_{vl}}\}$ (shown in Figure 1). We model the OBS node that the supercomputer is connected to as d , where all the files are destined to.

The objective is to determine the following:

- 1) *Route*: The path through which a file should be transferred from the source to the destination.
- 2) *Time schedule*: The time at which a file has to be transmitted in a single burst so that it can be transferred through the route determined in Step 1. This is important because two files which share a link on their routes should not be transmitted at the same time to avoid collision due to the constraints of an OBS network described below.

In an OBS network, although limited data buffering at OBS nodes is currently possible using fiber delay lines, it is inadequate for buffering very large files as they exist in our case. Hence, once a data warehouse starts transmitting a file, it must reach the destination in a single burst. We assume that the files cannot be fragmented. This simplifies the burst-assembly process and reduces the overhead of burst regeneration at the destination.

The aim is to minimize the total time for data aggregation. This is assuming that the last file to reach the destination is indeed the bottleneck, since computation cannot begin unless all the data is accumulated.

The two dimensions of determining both the path and the time makes this problem exceptionally hard, and differentiates it from all machine-scheduling problems which have been reported in the literature [4].

We call the above problem the Time-Path Scheduling Problem (TPSP). In the following sections, we shall present various approaches to solve TPSP.

III. NP-COMPLETENESS OF TPSP

We first model the optimization TPSP as a decision TPSP, by asking if TPSP can be solved within a deadline D .

Clearly, the problem is in *NP* because, given a solution, it is easy to verify if the last file reaches the destination within D , and if any constraints are violated.

We present a proof for polynomial reduction of the multiprocessor-scheduling problem (MSP) [5], which is known to be NP-complete, to TPSP.

Multiprocessor Scheduling [5]: Given a set of T tasks, and a number $m \in \mathbb{Z}^+$ of processors, length $l(t) \in \mathbb{Z}^+$ for each task $t \in T$, and a deadline $D \in \mathbb{Z}^+$, is there a schedule that meets the deadline D given that no two tasks can be processed in the same processor at the same time?

MSP may be reduced to TPSP by constructing the following graph $G(V, E)$.

- 1) Construct a vertex for each processor. Thus, we have m vertices labelled $1, 2, \dots, m$.
- 2) Construct a vertex for the destination node. Call this d . Construct one edge from each of the vertices $1, 2, \dots, m$ to d .
- 3) Construct a dummy vertex for source. Call this s . Construct an edge from s to each of the vertices $1, 2, \dots, m$.
- 4) Model all the tasks $t \in T$ as files whose transfer time T_f is the same as the length of the tasks $l(t)$. Node s will be the source node for all these files.

Now, TPSP is formulated as follows. Does there exist a time-path schedule through which the files at node s can be transferred to destination d within time D ?

We shall now prove that MSP has a solution if and only if TPSP has a solution. Suppose MSP has a solution. Consider a task t_k which is scheduled at machine p from time τ_k to time $\tau_k + l(t_k)$. This can be scheduled on the path $s-p-d$ from time τ_k to time $\tau_k + l(t_k)$ in TPSP. Since the machines are guaranteed to process only one task at a time, it is guaranteed that the path $s-p-d$ will transfer only one file at a time. Since MSP gives solution within deadline D , it is guaranteed that TPSP will have a solution within the deadline D . Now, let TPSP have a solution. Then, all files are transferred along one of the paths $s-p-d$ where $p \in 1, 2, \dots, m$. Each of these paths may be modelled to one machine in MSP. If the path of two files shares a common link in TPSP, they cannot be scheduled at the same time. This guarantees that one processor is not processing two tasks simultaneously in MSP. Thus, a solution in TPSP has a solution in MSP. We thus prove that MSP is polynomial-time reducible to TPSP. This proves the NP-completeness of TPSP.

IV. MATHEMATICAL MODEL OF TPSP

We formulate TPSP as an optimization problem based on the concepts of virtual-topology design in optical networks [6].

Given:

- 1) Set R of OBS nodes in the network.
- 2) The destination OBS node at which the supercomputer is located, d .
- 3) Set M of files which have to be transferred to the destination d .
- 4) Physical-connectivity adjacency matrix, $P(i, j), \forall i, j \in R$. $P(i, j)$ takes two values, 0 and 1. $P(i, j) = 1$ denotes connectivity.

- 5) OBS nodes at which the files are located (and to which the corresponding data warehouses are connected), $N(m) \in R$, $\forall m \in M$.
- 6) Transfer Time (T_f) for each file, $T(m) \in Z^+$, $\forall m \in M$. This can be pre-computed using Equation (1).

Subject variables:

- 1) Virtual-connectivity matrix, $V_{i,j}^m$, $\forall i, j \in R, m \in M$, takes two values, 0 and 1. $V_{i,j}^m = 1$ denotes that the file m is routed along a path which contains the link from i to j .
- 2) Start time $\tau(m)$, $\forall m \in M$, denotes the time at which the file m is transmitted. File m is transferred along the determined route from time $\tau(m)$ till time $\tau(m) + T(m)$.

Constraints:

- 1) *Connectivity constraints:* These constraints ensure proper virtual connectivity.

$$V_{i,j}^m \leq P_{i,j} \quad \forall i, j \in R, m \in M \quad (2)$$

$$\sum_{j=1}^{j=|R|} V_{N(m),j}^m = 1 \quad \forall m \in M \quad (3)$$

$$\sum_{m=1}^{m=|M|} \sum_{j=1}^{j=|R|} V_{j,d}^m = |M| \quad (4)$$

$$\sum_{j=1}^{j=|R|} V_{x,j}^m = 1 \quad \forall x \in R, \forall m \in M \quad (5)$$

$$\sum_{j=1}^{j=|R|} V_{j,x}^m = \sum_{k=1}^{k=|R|} V_{x,k}^m \quad \forall x \in R, m \in M \quad (6)$$

Explanation of equations: Constraint (2) ensures that a virtual link may exist only if a physical link exists. Constraint (3) ensures that a virtual link must exist from the source node of each file to the next node. Constraint (4) ensures that the destination must have one incoming virtual link for each file. Constraint (5) ensures that there is no bifurcation in the path for a particular file. Constraint (6) is flow-constraint equation for balanced flows. The number of incoming virtual links at a node for a particular file should equal the number of outgoing virtual links for that file.

- 2) *No time-overlap constraints:* These constraints ensure that, if a link is utilized for transferring one file, then it can be used for another file only after or before the file has been transmitted, but not during. At least one of the following three constraints must be satisfied. For any link (i, j) and pair of files (m, m') :

$$V_{i,j}^m + V_{i,j}^{m'} \leq 1 \quad (7)$$

$$\tau(m') \geq \tau(m) + T(m) \quad (8)$$

$$\tau(m) \geq \tau(m') + T(m') \quad (9)$$

Explanation of equations: Constraint (7) implies that link (i, j) is not shared by the two files (m, m') . Constraint (8) implies that link (i, j) is used for transferring file m' only after file m has been transferred. Constraint (9) implies that link (i, j) is used for transferring file m only after file m' has been transferred.

- 3) *Subject variable constraints:*

$$\tau(m) \geq 0 \quad \forall m \in M \quad (10)$$

Objective function:

$$\text{Minimize}(\text{Max}(\tau(m) + T(m))) \quad \forall m \in M \quad (11)$$

The objective function aims at minimizing the time at which the last file is received at the destination, hereafter called the *finish time*.

The *no time-overlap constraints* and *objective function* can be easily represented as linear equations by introducing some dummy integer variables. Variables $V_{i,j}^m$ are constrained to be integers, while $\tau(m)$ is real. Therefore, the formulation turns out to be a Mixed Integer Linear Program (MILP), which can be solved using a MILP solver [7].

The size of the MILP grows rapidly with the number of files because a set of several equations is created for every pair of files. Therefore, we propose efficient heuristics to solve the problem, and we use the MILP for only a comparative study.

Lower bound on finish time: Let the number of optical links through which destination d is connected to its neighbouring OBS nodes be l . Since only one file may be transferred along a link at a time, d may receive only l files simultaneously. Thus, a lower bound on the *finish time* may be stated as:

$$T_{fin}^{lb} = \frac{\sum T_f}{l} \quad (12)$$

V. HEURISTICS

We propose three heuristics to yield close-to-optimum solutions for TPSP. For analysis of the worst-case running-time complexity, we denote r as the number of OBS nodes, and f as the number of files.

LONGEST-FILE-FIRST (LFF) SCHEDULING:

This heuristic is based on the intuition that the longest file (having the largest transfer time) is the bottleneck for scheduling, because it requires more resources in terms of the amount of time required to be free on the links to be transferred. The LFF algorithm aims at scheduling the longest files first so that they get the priority to be scheduled earlier. For choosing the path over which to transfer a file, the algorithm chooses the best path among K randomly chosen paths. The steps are:

1. Choose the longest file F which has not yet been scheduled.
2. Find random K - alternate paths from the source node of file F to destination d . Random K - alternate paths may be achieved by randomly picking the weights of the links and applying Dijkstra's algorithm to compute shortest path [8].

3. Out of these K paths, find one in which file F can be scheduled at the earliest.
4. Repeat Step 1 until all files are scheduled.

Running-time complexity: Step 3 has running time of $O(Krf)$ in the worst case because any chosen path may have length $O(r)$ and $O(f)$ files already scheduled on it. Since Steps 1...4 are repeated for each file, the overall worst-case running-time complexity of LFF is $O(Krf^2)$.

DISJOINT-PATH (DP) SCHEDULING:

This heuristic is based on the intuition that files can be transferred along link-disjoint paths in parallel. The idea is to compute the maximum number of disjoint paths from the sources of the files to destination d . The above can be computed through an implementation of the Max-Flow algorithm [8] on the following modified graph. All the links have unit capacity. A dummy source node is connected to all the nodes which have files not scheduled as yet, with link capacity as the number of files. The destination is connected to a dummy destination with capacity as the number of files yet to be scheduled. The Max-Flow algorithm then identifies the disjoint paths to consist of links with unit flow. The steps of the algorithm are:

1. Let $StartTime = 0$. $StartTime$ is the time at which scheduling starts for a particular iteration.
2. Calculate the disjoint paths as described above.
3. Calculate the earliest completion time by which all the files at any node can be scheduled along the disjoint paths originating from it. Let this time be denoted as T , and the corresponding node as x . Therefore, by time T , all the files at node x can be scheduled. Now, for all remaining nodes for which disjoint paths originate, schedule as many files as possible in time T , starting from the longest file. All these files are scheduled in the time interval $StartTime$ to $StartTime + T$.
4. After Step 3, we have at least one node all of whose files have been routed. Increment $StartTime$ to $StartTime + T$. Now, repeat Step 2 till all files at all nodes have been scheduled.

Running-time complexity: Step 2 has worst-case time complexity of $O(r^3)$ [8]. Step 3 has worst-case time complexity of $O(f)$, since all the files must be traversed. During each iteration, at least one node has all files scheduled; hence, the maximum number of iterations of Steps 2...4 is $O(r)$. Therefore, the worst-case running-time complexity of the DP heuristic is $O(r^4 + rf)$.

MOST-DISTANT-FILE-FIRST (MDFF) SCHEDULING:

This heuristic is a slight variant of LFF. The idea here is that files which are most distant in terms of number of hops from the destination occupy more links and are hence the bottleneck for scheduling. The heuristic aims at scheduling these files first when the network is relatively resource-free. The remaining algorithm and the worst-case running-time complexity are on

the lines of LFF.

Analysis of time complexities: Both LFF and MDFF have the same worst-case time complexity. DP has better time complexity compared to LFF and MDFF, when $f > r^{3/2}$. So, whenever the number of files is large, we expect DP to perform faster.

VI. ILLUSTRATIVE NUMERICAL RESULTS

To compare the performance of our heuristic algorithms, we simulate them on a Java-based simulator, on the 24-node sample US nationwide network topology shown in Figure 2. All link capacities are $C = OC-192$ (10 Gbps). Node 3 is selected as the destination node where the supercomputer exists. It is assumed that a specified number of files of sizes randomly distributed between 5 Gbytes and 20 Gbytes are located randomly across the remaining nodes in the network. Equation (1) is used to calculate the transfer time for each file. The maximum propagation delay, P_d , is taken to be 25 ms which is typically the delay encountered in a 2500-km long fiber link. The maximum control overhead, O_c , is assumed to be 1 ms. In the simulations, we increase the number of files gradually from 50 to 300. For each setting, we measure the finish time. Figure 3 shows the results from LFF for different values of the number of alternate paths K . As expected, the algorithm performs increasingly better as K is increased from 2 to 5, because of increased choices for scheduling. Our choice

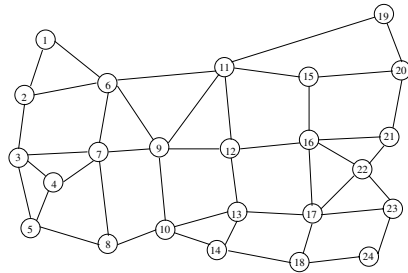


Fig. 2. 24-node sample network topology chosen for our simulation experiments.

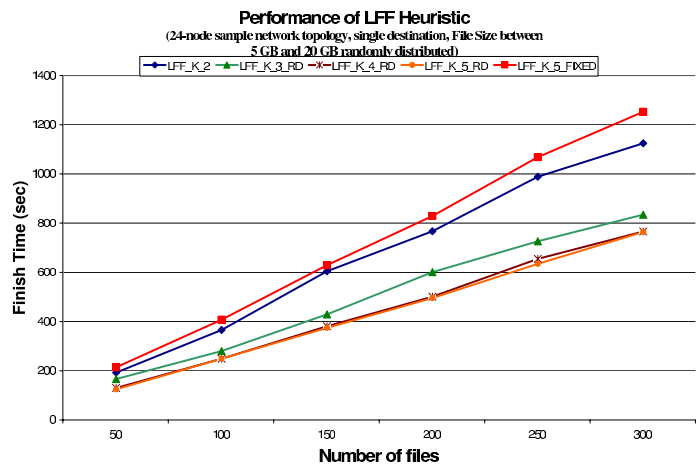


Fig. 3. Results from LFF heuristic for 24-node topology.

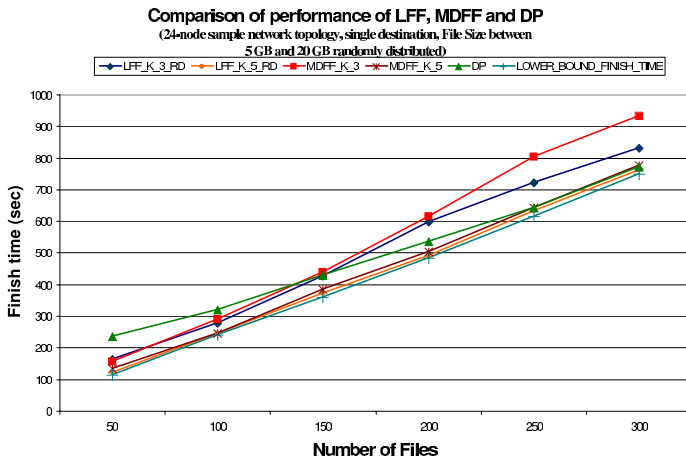


Fig. 4. Comparison of the three heuristics – LFF, MDFF, and DP – with lower bound on *finish time*.

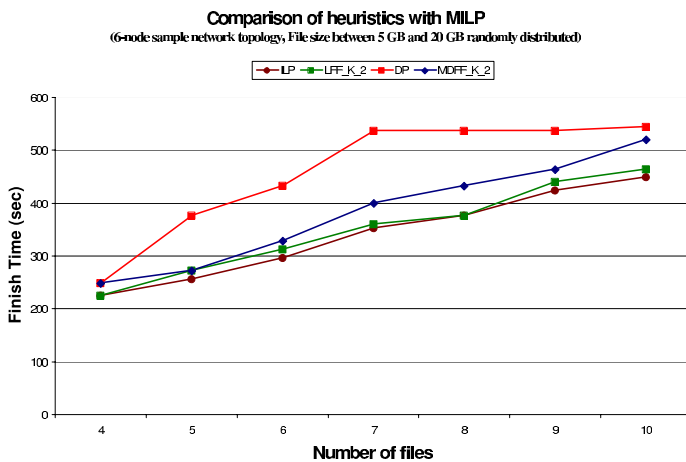


Fig. 5. Comparison of heuristics with MILP.

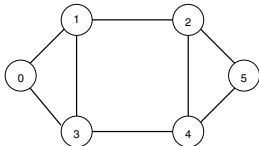


Fig. 6. 6-node sample network topology chosen for comparing our heuristics with our MILP.

of computing random paths is justified, as we find that the algorithm performs considerably poorly when paths are fixed (LFF_K_5_FIXED). This is because the fixed set of paths gets increasingly congested as more files originating from the same source node are scheduled along the same fixed set of routes. We observed a similar trend in MDFF (not shown here) as K is increased from 2 to 5.

Figure 4 compares the performance of the three heuristics. Also plotted is the lower bound on *finish time* calculated from Equation (12). We have plotted the performance of LFF and MDFF for values of $K = 3$ and $K = 5$. We observe that LFF performs slightly better than MDFF. The difference is more perceptible when the number of files is larger. An interesting

trend is the performance of DP. For fewer files, DP does not perform well compared to LFF or MDFF. However, with increasing number of files, DP’s performance also improves, and eventually it outperforms both MDFF and LFF with $K = 3$ and performs very close to LFF with $K = 5$. This is attributed to the following reason. In Step 3 of DP, files from nodes other than node x are scheduled in time T . If no file at a particular node is small enough to be scheduled in this time, then this time T remains unutilized. As the number of files is increased, it creates diversity in the sizes of the files at all nodes. The size diversity helps in packing files in time T more efficiently. Hence, the amount of unutilized time is decreased. We also note that DP has better running-time complexity than LFF or MDFF for large number of files. Therefore, we conclude that, while LFF should be the preferred heuristic with smaller number of files, DP should be chosen for larger number of files. We also observe that all the heuristics perform close to the lower bound on *finish time*. This confirms that the heuristics are performing efficiently.

Finally, we compare the performance of the heuristics against optimal values achieved from the MILP using a CPLEX MILP solver [7]. Since the MILP does not scale well with the size of the input, we use the small 6-node topology shown in Figure 6. Node 1 is chosen as the destination. The number of files is varied from 4 to 10. Figure 5 compares the MILP solutions with the ones achieved by our heuristics. We observe that LFF performs very close to the optimal solution. DP performs considerably poorly because the number of files is quite low, and this is consistent with the reasoning provided earlier. We showed earlier that, for large number of files, DP performs very close to LFF with $K = 5$. Therefore, we expect both DP and LFF to give solutions very close to optimal values for large number of files.

VII. CONCLUSION

In this work, we have defined the Time-Path Scheduling Problem (TPSP) for aggregating large data files from distributed data warehouses. We proved that the problem is NP-complete. We then presented an MILP-based solution and three heuristics: LFF, DP, and MDFF. We observed that, while LFF should be preferred for a small number of files, DP should be preferred for scheduling a larger number of files.

REFERENCES

- [1] “Genomes To Life requires new life from Networks,” Department of Energy (DoE) (United States) Workshop, 2003, <http://www.csm.ornl.gov/ghpn/genome.wk2003.pdf>
- [2] M. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, Second Edition, Prentice-Hall, 1999.
- [3] T. Battestilli and H. Perros, “An introduction to optical burst switching,” *IEEE Communications Magazine*, vol. 41, no. 8, August 2003.
- [4] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Second Edition, Prentice Hall, 2002.
- [5] M. Garey and D. Johnson, *Computers and Intractability: A guide to the theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
- [6] B. Mukherjee, *Optical Communication Networks*, McGraw-Hill, pp. 259–288, 1997.
- [7] <http://www.ilog.com/products/cplex/product/suite.cfm>
- [8] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, “Introduction to Algorithms,” Second Edition, MIT Press, 2001.