A Tight Approximation for Co-flow Scheduling for Minimizing Total Weighted Completion Time

Sungjin Im*  Manish Purohit†

Abstract

Co-flows model a modern scheduling setting that is commonly found in a variety of applications in distributed and cloud computing. In co-flow scheduling, there are \( m \) input ports and \( m \) output ports. Each co-flow \( j \in J \) can be represented by a bipartite graph between the input and output ports, where each edge \((i, o)\) with demand \( d_{i,o}^j \) means that \( d_{i,o}^j \) units of packets must be delivered from port \( i \) to port \( o \). To complete co-flow \( j \), we must satisfy all of its demands. Due to capacity constraints, a port can only transmit (or receive) one unit of data in unit time. A feasible schedule at each time \( t \) must therefore be a bipartite matching.

We consider co-flow scheduling and seek to optimize the popular objective of total weighted completion time. Our main result is a \((2 + \epsilon)\)-approximation for this problem, which is essentially tight, as the problem is hard to approximate within a factor of \((2 - \epsilon)\). This improves upon the previous best known 4-approximation. Further, our result holds even when jobs have release times without any loss in the approximation guarantee. The key idea of our approach is to construct a continuous-time schedule using a configuration linear program and interpret each job’s completion time therein as the job’s deadline. The continuous-time schedule serves as a witness schedule meeting the discovered deadlines, which allows us to reduce the problem to a deadline-constrained scheduling problem.

*Electrical Engineering and Computer Science, University of California, 5200 N. Lake Road, Merced CA 95344. sim3@ucmerced.edu. Supported in part by NSF grant CCF-1409130 and CCF-1617653.
†Google, Mountain View, CA
1 Introduction

Co-flow scheduling [4] is an elegant scheduling model that abstracts a signature scheduling problem that characterizes modern massively parallel computing platforms such as MapReduce [7] and Spark [22]. Such platforms have a unique processing pattern that interleaves local computation with communication across machines. Due to the size of the large data sets processed, communication typically tends to be the main bottleneck in the system performance of these platforms, and the co-flow model captures the key scheduling challenge arising from such communication.

In co-flow scheduling, there are $m$ input ports and $m$ output ports. Each job (or co-flow) $j$ is described by a bipartite graph between input and output ports where each edge $(i, o)$ is associated with a demand $d_{i,o}^j$, meaning the job has $d_{i,o}^j$ unit-sized packets to send from input port $i$ to output port $j$. Job $j$ completes at the earliest time when all its packets are delivered. We assume that time is slotted and at each integer time $t$, a feasible schedule is a matching, and exactly one packet of a certain co-flow is delivered from input port $i$ to output port $o$ when $i$ is matched to $o$. At a high-level, a packet delivery is data migration from a machine $i$ to another machine $o$ before starting the next round of local computation. Co-flow scheduling can also be viewed as a generalization of the classical concurrent open shop scheduling problem [2, 3, 8, 13, 15, 18, 21], as we can recover the concurrent open shop when $d_{i,o}^j = 0$ for all jobs $j$, and pairs $i \neq o$ of ports.

Since co-flow scheduling is an abstraction of one-round communication which is potentially part of multiple rounds of computation, some packets become ready to be transferred only later in the schedule. This necessitates the study of co-flow scheduling with packet release times. Thus, a job $j$ may have a release/arrival time, $r_j$, meaning that no packet of job $j$ can be transported before its release time. More generally, we can consider a different release time for each packet of a job and all our results extend naturally to this setting. For simplicity, we assume that all packets of job $j$ have the same release time $r_j$.

In this paper we study co-flow scheduling to minimize the (weighted) average / total completion time, one of the most popular objectives used to measure average job latency. Not surprisingly, being at the heart of modern parallel computing, co-flow scheduling has been actively studied by both the system and theory communities [1, 4, 5, 6, 12, 14, 16, 10, 20, 23], particularly for the completion time objective [1, 10, 11, 12, 14, 16, 20]. Since even the special case of concurrent open shop scheduling is hard to approximate within factor $2 - \epsilon$ [2, 18], there has been a sequence of attempts to give approximate scheduling algorithms. Prior to this work, the best known approximation ratio were 5 for the general case, and 4 when all jobs arrive at time 0 [1, 20], respectively. Thus, it remained open to reduce the gap between best upper and lower bounds.

1.1 Our Result

In this paper, we answer the open problem in the affirmative by giving an essentially tight approximation algorithm.

**Theorem 1.1.** There is a deterministic 2-approximation algorithm for co-flow scheduling for minimizing total weighted completion time when all parameters such as demands and release times are polynomially bounded by the input size. For the general case, there is a $(2 + \epsilon)$-approximation for any constant $\epsilon > 0$.

This approximation guarantee is almost tight as there is an inapproximability result of $2 - \epsilon$ [2, 18]. Thus, our result essentially closes the approximability of co-flow scheduling for the completion time minimization objective. Interestingly, we have the same approximation guarantee for general co-flow scheduling with arbitrary release times as for the special case when all jobs are released at time 0.

**Other Extensions.** As mentioned before, our result can be easily extended to handle the case where packets of the same job have different release times. As pointed out in the seminal paper [4], in practice, ports can have non-uniform capacities, meaning that each port can route a different number of packets in each time slot. Our algorithm easily generalizes to handle ports with non-uniform capacities (see Section [7] for a sketch).
Interestingly, even for the classical concurrent open shop scheduling with release times, the previous best known approximation factor was $3 \ [1, 8, 13]$. Theorem 1.1 immediately yields an improved approximation algorithm for preemptive concurrent open shop with arbitrary release times.

**Corollary 1.2.** There is a deterministic 2-approximation algorithm for preemptive concurrent open shop scheduling with release times when all input parameters such as processing times and release times are polynomially bounded. For the general case, there is a $(2 + \epsilon)$-approximation algorithm for any constant $\epsilon > 0$.

1.2 Our Techniques

We first discuss approaches used in the previous work [16, 1] along with their limitations. We then give a high-level overview of our approach and highlight our key ideas that enable an essentially tight approximation. To streamline our discussion, let’s assume for a while that all jobs arrive at time 0.

**Previous Approaches.** We first discuss the work by Qiu et al. [16], which gave the first non-trivial approximation for the completion time objective. A key observation therein is that, individually, each job $j$ can be scheduled within $\Delta(j)$ time units if there were no other jobs where $\Delta(j)$ denotes the maximum degree of the bipartite graph representing job $j$. This idea is easily extended to multiple jobs by aggregating them into one job. More precisely, a subset $S$ of jobs can be completed within $\Delta(S)$ time steps where $\Delta(S)$ is the maximum degree of the bipartite graph that adds up all demands of each edge over the jobs in $S$. They set up an interval-indexed linear programming relaxation to find a tentative completion time for each job. While rounding, jobs with similar tentative completion times are grouped together and scheduled separately in $\Delta(S)$ time steps (where $S$ denotes the set of jobs grouped together). The disjoint schedules for each group are then concatenated sequentially. Since jobs in a group $S$ are scheduled independently of other groups so that all jobs finish by time $\Delta(S)$, their approach can be viewed as a reduction to the makespan objective where jobs have a uniform deadline. Unfortunately, such an approach cannot give a tight approximation as a result of concatenation of disjoint schedules. Using these techniques, Qiu et al. give a randomized 16.54-approximation algorithm.

The work by Ahmadi et al. [1] giving the best known 4-approximation improves upon Qiu et al.’s result by a reduction to the Concurrent Open Shop problem (COS). In the COS problem, there are $m$ machines. Each job has a certain workload on each machine and completes when all its workload has been completed. Thus, the COS does not have to cope with the complexities coming from the bipartite graph structure as in co-flow scheduling. Their LP relaxation adapts the LP relaxation for COS by pretending that each port is an individual machine. Their approach loses a factor of 2 in the rounding just as the LP rounding for the COS problem does. An additional factor 2 loss follows since the same rounding for the COS does not yield a feasible schedule for co-flow as the LP was set up without fully capturing the interaction between input and output ports. Their approach crucially uses non-uniform tentative completion times for the improvement upon the result by Qiu et al. [16], but fails to give a tight approximation by failing to effectively capture the underlying graph structure.

**Our Approach.** To get a tight approximation, we seek to use non-uniform deadlines as in [1] but at the same time ensure that we fully factor in the underlying graph structure. At a high level, our approach can be summarized as a true reduction of the problem of minimizing total completion time to a deadline-constrained scheduling problem in the following sense: We find a tentative completion time / deadline $C^*_j$ for each job $j$ such that (i) it is guaranteed that there exists a schedule where every job $j$ completes before its deadline $C^*_j$, and (ii) any deadline-meeting schedule is 2-approximate for the total completion time objective.

Towards this end, we use a configuration LP where we create a variable for each job’s each possible schedule pretending that no other jobs exist; note that each variable completely determines the job’s schedu-
ule, particularly its completion time. The configuration LP has exponentially many variables but can be solved by solving its dual using a separation oracle. The separation oracle boils down to finding the cheapest schedule for a given job when ports are priced differently over time, which can be solved using network flow. The configuration LP gives a fractional solution with a nice property: when a job completes by $\lambda$ fraction by time $t$, $\lambda$ fraction of the job’s complete schedules are packed by that time $t$. Using this property, one can find a fractional schedule with deadlines $C_j^*$ giving the desired properties, (i) and (ii)—in particular, unit fraction of each job’s schedules is packed by $C_j^*$, which implies a witness fractional flow guaranteeing (i). Then, we again use network flow to find an actual schedule.

Our approach is readily extended to factor in job release times without sacrificing the approximation quality. In fact, thanks to our simple and general approach, we can easily extend our result to capture different release times even at the packet level as well as to capture non-uniform port capacities. We believe our high-level approach that determines ‘tight’ non-uniform deadlines for jobs admitting a feasible schedule by constructing a witness fractional schedule, could find more applications in other scheduling contexts.

1.3 Other Related Work

As mentioned before, the Concurrent Open Shop problem (COS) [2, 18, 8, 21, 15, 3, 13] is a special case of the co-flow scheduling problem. However, in the COS, preemption is typically disallowed, meaning that any task started on a machine must be processed until its completion without interruption once it gets started. It is easy to see that preemption does not help if all jobs arrive at time 0, in which case several 2-approximation algorithms were shown [3, 8, 13] via LP rounding, which were later shown to be tight [2, 18]. When jobs have different release times, the same LP relaxations yielded 3-approximations [8, 13]. Later, Mastrolilli et al. [15] gave a simple greedy algorithm that matches the best approximation ratio when all jobs arrive at time 0. Recently, Ahmadi et al. [1] gave a combinatorial 3-approximation via a primal-dual analysis when jobs have non-uniform release times.

2 Problem Definition and Notation

Following the approach taken by all prior work [1, 6, 16], we abstract out the network as a single $m \times m$ non-blocking switch with unit capacity constraints, i.e., any input (or output) port can only transmit (or receive) one unit of data at any time slot. Let $m$ denote the number of input and output ports in the system and let $J$ denote the set of all co-flows. As jobs are the more commonly used terminology in scheduling literature, we may refer to co-flows as jobs. Each job $j \in J$ is represented as a bipartite graph $G^j$ between the input and output ports where each edge $(i, o)$ has a weight $d_{io}^j$ that represents the number of unit sized packets that co-flow $j$ needs to deliver from port $i$ to port $o$. Each job $j$ also has a weight $w_j$ that indicates its relative importance and a release time $r_j$. A co-flow $j$ is available to be scheduled at its release time $r_j$ and is said to be completed when all the flows of the job $j$ have been scheduled. More formally, the completion time $C_j$ of co-flow $j$ is defined as the earliest time such that for every input port $i$ and output port $o$, $d_{io}^j$ units of its data have been transferred from port $i$ to port $o$. We assume that time is slotted and data transfer within the network is instantaneous. Time slots are indexed by positive integers. By time slot $t$, we refer to time interval $[t-1, t)$ of unit length. Since each input port $i$ can transmit at most one unit of data and each output port $o$ can receive at most one unit of data in each time slot, a feasible schedule for a single time slot can be described as a matching. Our goal is to find a feasible schedule that minimizes the total, weighted completion time of the co-flows, i.e. minimize $\sum_j w_j C_j$.

3 Linear Programming

3.1 A Simplifying Assumption

In this section, we present our configuration linear program. To make our presentation more transparent, we will first assume that the maximum potential completion time $T$ of all jobs is polynomially bounded.
by the input size. Note that this simplifying assumption holds true when all parameters, particularly job
demands and release times are polynomially bounded. The main challenges still remain the same under this
assumption, which will be removed in Section 5.

### 3.2 LP Formulation

We formulate a configuration linear program. For a job $j$, a configuration is a complete feasible, integral
schedule for $j$. More formally, a configuration $F$ for job $j$ is a ternary relation of tuples of the form $(i, o, t)$
that indicates one unit of data is transferred from input port $i$ to output port $o$ at time $t$. Since $F$ is a feasible
configuration for $j$, by definition, the relation must satisfy:

1. $|\{(i,o,t) \in F\}| = d_{io}^j$ for all $i, o \in [m]$.
2. $|\{(i,o,t) \in F\}| \leq 1$ for each pair of $i \in [m]$ and $t \geq 1$.
3. $|\{(i,o,t) \in F\}| \leq 1$ for each pair of $o \in [m]$ and $t \geq 1$.

Let $\mathcal{F}(j)$ denote the set of all possible feasible schedules of job $j$. Let $C_j^F$ denote the completion time
of job $j$ under schedule $F \in \mathcal{F}(j)$. In other words $C_j^F = \max \{t \mid \exists i \in [m], o \in [m] \text{ s.t. } (i, o, t) \in F\}$.

We introduce variables $\{y_j^F\}$ to indicate whether job $j$ is scheduled as per configuration $F$. The following
shows our linear programming formulation, which we refer to as LP$_{\text{primal}}$.

$$\text{LP}_{\text{primal}} : \quad \min \sum_{j \in J} \sum_{F \in \mathcal{F}(j)} w_j \sum_{j \in J} C_j^F y_j^F$$

subject to,

\begin{align}
\forall j \in J, \quad & \sum_{F \in \mathcal{F}(j)} y_j^F \geq 1 \quad (1) \\
\forall i \in [m] \text{ and } \forall t \in [T], \quad & \sum_{j \in J, o \in [m]} \sum_{F \in \mathcal{F}(j) \text{ s.t. } (i, o, t) \in F} y_j^F \leq 1 \quad (2) \\
\forall o \in [m] \text{ and } \forall t \in [T], \quad & \sum_{j \in J, i \in [m]} \sum_{F \in \mathcal{F}(j) \text{ s.t. } (i, o, t) \in F} y_j^F \leq 1 \quad (3) \\
\forall j \in J \text{ and } \forall F \in \mathcal{F}(j), \quad & y_j^F \geq 0 \quad (4)
\end{align}

In the objective, each job $j$’s weighted completion time is $w_j C_j^F$ under configuration $F \in \mathcal{F}(j)$. Constraint (1) ensures that every co-flow is scheduled as per some configuration. Constraints (2) and (3) ensure that at any time step $t$ at most one co-flow uses an input port $i$ or output port $o$. In constraint (4), we relax the integrality requirements to get a linear programming relaxation.

### 3.3 Solving the Linear Program

The linear programming relaxation LP$_{\text{primal}}$ has exponentially many variables but polynomially many con-
straints (under the simplifying assumption that $T$ is polynomially bounded). To solve LP$_{\text{primal}}$, we formulate
its dual, LP$_{\text{dual}}$, which has polynomially many variables but exponentially many constraints, and demon-
strate a polynomial-time separation oracle for LP$_{\text{dual}}$. Using the Ellipsoid method with the separation oracle,
we can solve LP$_{\text{dual}}$ in polynomial time. During the execution of the Ellipsoid method, only polynomially
many constraints of the dual are considered, thus, as a result of the strong duality theorem, we only need to
consider the corresponding variables to optimally solve LP$_{\text{primal}}$. Therefore, a polynomial-time separation
oracle for LP$_{\text{dual}}$ implies that LP$_{\text{primal}}$ can be solved in polynomial time.

The following shows the dual of LP$_{\text{primal}}$. 


\[
\text{LP}_{\text{dual}} : \quad \max \sum_{j \in J} \alpha_j - \sum_{i \in [m], t \in [T]} \beta_{i,t} - \sum_{o \in [m], t \in [T]} \gamma_{o,t}
\]
subject to, \quad \forall j \in J and \forall F \in \mathcal{F}(j),
\[
\alpha_j - \sum_{(i,o,t) \in F} (\beta_{i,t} + \gamma_{o,t}) \leq w_j C_j^F 
\tag{5}
\]
\[
\alpha, \beta, \gamma \geq 0
\]

It now remains to show a separation oracle for \( \text{LP}_{\text{dual}} \). Our task is, given \( \alpha_j, \beta_{i,t} \) and \( \gamma_{o,t} \) values, to find a job \( j \) and a configuration \( F \in \mathcal{F}(j) \) that violates constraints \( 5 \) or certify that no such constraint exists. We observe that such a separation oracle is reduced to solving the following key problem.

**Key Problem for the Separation Oracle:** Given a single job \( j \) and a deadline \( C_j \in [T] \), find a feasible, integral schedule for \( j \) of minimum cost when input port \( i \) is priced at \( \beta_{i,t} \) at time \( t \) and output port \( o \) is priced at \( \gamma_{o,t} \) at the same time.

The observation immediately follows by considering a fixed job \( j \) and a fixed completion time \( C_j \in [T] \) — if the cost of the job’s minimum-cost deadline-constrained schedule \( F^* \) is less expensive than \( \alpha_j - w_j C_j^F \), then we report the corresponding configuration as a violated constraint. Otherwise, if the minimum cost schedules for all deadlines satisfy the constraints, then we are guaranteed that no constraints are violated. Suppose for contradiction that some configuration \( F \) violates constraint \( 5 \). Let \( F^* \) be the minimum cost schedule with deadline \( C_j^F \). Thus, we have \( \text{cost}(F^*) \leq \text{cost}(F) < \alpha_j - w_j C_j^F \leq \alpha_j - w_j C_j^{F^*} \), which is a contradiction.

We show how to reduce this key problem to an instance of the classical minimum cost flow problem. Create a directed graph \( G \) as follows. For each \( t \in [C_j] \setminus [r_j] \), create a complete, bipartite, directed subgraph \( G_t = (I_t, O_t, A_t) \) where part \( I_t \) has \( m \) nodes corresponding to the input ports, \( O_t \) has \( m \) nodes corresponding to the output ports. For an edge \( e = (i_t, o_t) \), assign a cost of \( \beta_{i,t} + \gamma_{o,t} \). Assign a capacity of one for all edges and all vertices in \( G_t \). In addition, for every pair \( i, o \in [m] \), create vertices \( \text{source}_{io} \) and \( \text{sink}_{io} \). Connect \( \text{source}_{io} \) to vertices \( i_t \) for all \( t \in [C_j] \setminus [r_j] \) and connect vertices \( o_t \) for all \( t \in [C_j] \setminus [r_j] \) to \( \text{sink}_{io} \) with unit capacity and zero cost arcs. Finally, add two new vertices \( \text{source} \) and \( \text{sink} \). For every pair \( i, o \in [m] \), add an arc of capacity \( d^i_\text{io} \) and cost zero from \( \text{source} \) to \( \text{source}_{io} \) and also from \( \text{sink}_{io} \) to \( \text{sink} \). See Figure 1 for an illustration of this construction.

![Figure 1: An example of graph construction for the minimum cost flow problem](image)

---

*Vertex capacities can be modeled by splitting the a vertex into two and adding a unit capacity edge connecting the split parts.*
Note that any integral maximum flow $f$ in $G$ corresponds to a feasible schedule $F$ for job $j$ with completion time at most $C_j$ and vice versa. In particular, for any pair $i,o \in [m]$, the maximum flow sends $d_{io}^j$ units of flow from source$_{io}$ to sink$_{io}$ through $d_{io}^j$ edge disjoint paths each of the form $(source_{io} \rightarrow i_t \rightarrow o_t \rightarrow sink_{io})$. The vertex capacities ensure that for any $t \in [C_j] \setminus [r_j]$, the edges of $G_t$ carrying flow form a matching. Thus the set of all triples $(i,o,t)$ such that the edge $(i_t,o_t)$ carries unit flow forms a feasible configuration (schedule) for $d_{io}^j$. LP Rounding by solving the minimum cost maximum flow problem on $G$ the feasible schedule $F$ be used by $(C_j)$ completion time at most $C_j$ which will be discussed in Section 4.2 in detail.

4 Rounding

In this section, we show how to round an optimal feasible solution to LP$_{\text{primal}}$ to obtain a 2-approximate integral schedule. Towards this end, we consider some intermediate schedules that are continuous. We say that a schedule is continuous if it schedules an integral schedule. Towards this end, we consider some intermediate schedules that are continuous.

Our rounding scheme consists of two main components. In Section 4.1, we first find a deadline $C_j^*$ for each job $j$ that admits a feasible integral schedule that is 2-approximate against the optimal LP$_{\text{primal}}$ cost. We guarantee the feasibility by constructing a continuous schedule that completes each job $j$ at time $t$ that a schedule is continuous if it schedules an integral schedule. The procedure considers the following four schedules in this order.

1. \( \{ y_j^F \} \): An optimal solution to LP$_{\text{primal}}$. We view this as a sequence of fractional matchings over integer time slots. The cost is measured as the solution’s LP objective, which is denoted as $\text{cost}(\text{LP}_{\text{primal}})$.

2. \( \sigma \): A continuous schedule constructed from \( \{ y_j^F \} \). The cost is measured as the schedule’s total weighted fractional completion time: job $j$ is alive at time $t$ by at most $(1 - v)$ fraction and incurs a cost of at most $(1 - v)$ at the time if $j$’s each demand has been served by at least $v$ fraction by the time. The total cost over all jobs is denoted as $\text{cost}(\sigma)$ and we will show $\text{cost}(\sigma) = \text{cost}(\text{LP}_{\text{primal}}) - (1/2) \sum_j w_j$.

3. \( \sigma^\lambda \): A continuous schedule obtained by ‘stretching’ $\sigma$. We stretch the schedule $\sigma$ horizontally by a factor of $1/\lambda$. $\lambda \in (0, 1)$ is to be chosen randomly but the random choice can be derandomized. The cost is measured integrally, meaning that $j$ completes when all its demands are served. We will show $\mathbb{E}[\text{cost}(\sigma^\lambda)] \leq 2 \text{cost}(\sigma) = 2 \text{cost}(\text{LP}_{\text{primal}}) - \sum_j w_j$.

4. \( \{ C_j^* \} \): A continuous schedule with integer deadlines. This schedule is essentially the same as $\sigma^\lambda$. In $\sigma^\lambda$, jobs may have fractional completion times. We round them up to their nearest integers, respectively. Then, the cost of any schedule that completes each job $j$ before its predefined completion time (deadline) is at most $\mathbb{E}[\sum_j w_j C_j^*] \leq \mathbb{E}[\text{cost}(\sigma^\lambda)] + \sum_j w_j \leq 2 \text{cost}(\text{LP}_{\text{primal}})$.

The second step of our rounding process will find an actual schedule that complete each job $j$ by $C_j^*$, which will be discussed in Section 4.2 in detail.
In the rest of this section, we show how we construct each of the above schedules and upper bound its cost.

**The First Schedule (\(\{y^F_j\}\))**: Let \(\{y^F_j\}\) be an optimal solution to LP\(_{\text{primal}}\). We use these \(y^F_j\) values to construct a sequence of fractional matchings over integer time slots. Let \(x^t_{io} = \sum_{F \in F(j):(i,o,t) \in F} y^F_j \) denote the fraction of data of job \(j\) that is sent from port \(i\) to port \(o\) in time slot \(t\) \(\in [T] \setminus [r_j]\). Since we started from a feasible LP solution, the fractional coflow schedule defined by the \(x^t_{io}\) variables schedules \(d^j_{io}\) amount of data for any pair \((i, o)\) of ports.

\[
\sum_{t \in [T]} x^t_{io} = \sum_{t \in [T]} \sum_{F \in F(j)} \sum_{(i,o,t) \in F} y^F_j = \sum_{F \in F(j)} \sum_{t \in [T]} \sum_{(i,o,t) \in F} y^F_j = \sum_{F \in F(j)} y^F_j d^j_{io} = d^j_{io}
\]

Also for any fixed time slot \(t\), the variables \(x^t_{io}\) describe a fractional matching, i.e.

\[
\sum_{o \in [m]} \sum_{j \in J} x^t_{io} = \sum_{j \in J} \sum_{o \in [m]} \sum_{F \in F(j)} \sum_{(i,o,t) \in F} y^F_j \geq 1, \forall i \in [m]
\]

and

\[
\sum_{i \in [m]} \sum_{j \in J} x^t_{io} = \sum_{j \in J} \sum_{i \in [m]} \sum_{F \in F(j)} \sum_{(i,o,t) \in F} y^F_j \geq 1, \forall o \in [m]
\]

Note that the optimal cost of LP\(_{\text{primal}}\) can be written as

\[
\text{COST}(\text{LP}_{\text{primal}}) := \sum_j w_j \sum_{F \in F(j)} C^F_j y^F_j = \sum_j w_j \sum_{C \geq 1} \sum_{F \in F(j): C^F_j = C} y^F_j
\]

**The Second Schedule (\(\sigma\))**: We construct a continuous schedule \(\sigma\) from \(\{y^F_j\}\). Recall that in a continuous schedule, an integral matching is scheduled at each instantaneous time. Consider any fixed time slot \(t \in [T]\). Let \(X^t\) denote the fractional matching given by \(\{y^F_j\}\). By the Birkhoff-von Neumann theorem, \(X^t\) can be decomposed into a polynomial number of integral matchings \(M^1_t, M^2_t, \ldots, M^\ell_t\) such that

\[
X^t = \sum_{i=1}^{\ell} \alpha^t_i M^i_t \quad \text{and} \quad \sum_{i=1}^{\ell} \alpha^t_i = 1
\]

Thus, we can convert the fractional schedule for time slot \(t\) into a continuous time schedule by scheduling each integral matching \(M^i_t\) for \(\alpha^t_i(\tau_2 - \tau_1)\) time during each infinitesimal time interval \([\tau_1, \tau_2) \in [t-1, t)\). In other words, the linear combination of integral matchings is ‘smeared’ across all instantaneous time during \([t-1, t)\). Let \(\sigma\) denote such a continuous schedule obtained by decomposing the fractional matching in each time slot \(t \in [T]\). For any time interval \([\tau_1, \tau_2]\), let \(\int_{\tau_1}^{\tau_2} \sigma_{io,j}(\tau) d\tau\) denote the amount of data of co-flow \(j\) that is sent from port \(i\) to port \(o\) in the schedule \(\sigma\). By construction, we have that

\[
\int_{\tau=t-1}^{t} \sigma_{io,j}(\tau) d\tau = x^t_{io}.
\]

To compute COST\((\sigma)\), we need to measure how much fraction of each job \(j\) is completed by time \(\tau\). Recall that in this measure, job \(j\) is said to be completed by \(v\) fraction at time \(\tau\), if for every pair \(i, o\) of ports, at least \(vd^j_{io}\) amount of data is transferred from \(i\) to \(o\) by time \(\tau\) in schedule \(\sigma\), i.e., \(\int_{\tau_1}^{\tau_2} \sigma_{io,j}(\tau') d\tau' \geq vd^j_{io}\). Let \(\tilde{C}_j(v)\) denote the first time when \(j\) is completed by \(v\) fraction in \(\sigma\). The total fractional completion time of job \(j\) in schedule \(\sigma\) is thus defined as \(\int_{v=0}^{1} \tilde{C}_j(v) dv\). Formally, we now want to upper bound

\[
\text{COST}(\sigma) := \sum_j w_j \int_{v=0}^{1} \tilde{C}_j(v) dv.
\]
Lemma 4.1. \( \text{COST}(\sigma) = \text{COST}(\text{LP}_{\text{primal}}) - \frac{1}{2}(\sum_{j \in J} w_j) \).

Proof. Consider a fixed job \( j \) and a configuration \( F \in \mathcal{F}(j) \) with \( y_j^{F} > 0 \). Let \( t = C_j^{F} \) denote the last time slot during which \( F \) schedules a matching. Let us compute the contribution of job \( j \) to \( \text{COST}(\sigma) \) due to configuration \( F \).

Let \( M \) denote the matching scheduled by \( F \) in this last time slot. The LP solution schedules matching \( M \) by a \( y_j^{F} \) fraction in time slot \( t \). Since we ’smeared’ the fractional matching in time slot \( t \) over continuous times in the interval \([t-1, t]\), our continuous schedule \( \sigma \) schedules matching \( M \) by a \( y_j^{F}d\tau \) fraction during any time interval \([\tau, \tau + d\tau]\) \([t-1, t]\). Thus \( j \)'s contribution to \( \text{COST}(\sigma) \) due to configuration \( F \in \mathcal{F}(j) \) is \( w_j \int_{\tau=t-1}^{t} y_j^{F}d\tau = w_j y_j^{F}(C_j^{F} - 1/2) \).

Summing over all jobs \( j \) and all configurations, we have

\[
\text{COST}(\sigma) = \sum_j w_j \int_{v=0}^{1} \tilde{C}_j(v)dv = \sum_j w_j \sum_{F \in \mathcal{F}(j)} y_j^{F}(C_j^{F} - 1/2) = \text{COST}(\text{LP}_{\text{primal}}) - \sum_j w_j/2
\]
as desired. \( \square \)

Finally, we note that we don’t have to construct \( \sigma \) explicitly, as we just need to find \( \tilde{C}_j(v) \) in polynomial time for given \( j \) and \( v \in [0, 1] \).

The Third Schedule (\( \sigma^\lambda \)) : We construct a new continuous schedule \( \sigma^\lambda \) from \( \sigma \) as follows. First, we choose a \( \lambda \in [0, 1] \) randomly drawn according to the probability density function \( f(v) = 2v \). Next, we “stretch” the schedule \( \sigma \) by a factor of \( 1/\lambda \). More precisely, if matching \( M \) is scheduled in \( \sigma \) during an infinitesimal interval \([\tau_1, \tau_2]\), the same matching is scheduled in \( \sigma^\lambda \) during \([\tau_1/\lambda, \tau_2/\lambda]\). This ‘stretching’ (also called slow-motion) idea was used in other scheduling contexts \([19, 17, 9]\). Note that jobs’ release times remain respected after the stretching.

Let \( C_{\sigma^\lambda}(j) \) denote the completion time of job \( j \) in the stretched schedule, i.e., \( C_{\sigma^\lambda}(j) \) is the earliest time such that \( d_{jio}^l \) amount of data has been transferred from port \( i \) to port \( o \) for all pairs \( i, o \in [m] \). We now measure the cost of this new schedule, \( \text{COST}(\sigma^\lambda) := \sum_j w_j C_{\sigma^\lambda}(j) \), and upper bound it.

Lemma 4.2. \( \mathbb{E}[\text{COST}(\sigma^\lambda)] = 2 \text{COST}(\sigma) \).

Proof. Recall that we define \( \tilde{C}_j(v) \) to be the earliest time when job \( j \) is completed by \( v \) fraction in \( \sigma \). It is easy to observe that for every job \( j \), \( C_{\sigma^\lambda}(j) \leq \tilde{C}_j(\lambda)/\lambda \). This is because, \( \sigma \) schedules at least a \( \lambda \) fraction of the demand for each pair \( i, o \) by time \( \tilde{C}_j(\lambda) \) and thus \( \sigma^\lambda \) schedules all the demands fully by time \( \tilde{C}_j(\lambda)/\lambda \).

\[
\mathbb{E}[\text{COST}(\sigma^\lambda)] = \sum_j w_j \mathbb{E}[C_{\sigma^\lambda}(j)] \leq \sum_j w_j \mathbb{E}[\tilde{C}_j(\lambda)/\lambda] = \sum_j w_j \int_{v=0}^{1} \tilde{C}_j(v)/v \cdot (2v)dv \\
\leq 2 \sum_j w_j \int_{v=0}^{1} \tilde{C}_j(v)dv = 2 \text{COST}(\sigma)
\]

\( \square \)

The Last Schedule (\( \{C_j^*\} \)) : In the previous continuous schedule, \( \sigma^\lambda \), every job \( j \) is completed by time \( C_{\sigma^\lambda}(j) \), which is not necessarily an integer. Since we will use this completion time as \( j \)'s deadline in the second rounding procedure to find an actual schedule, we need to make sure the completion times / deadlines are integers. Hence, we set \( C_j := \lceil C_{\sigma^\lambda}(j) \rceil \). Then, we immediately have

\[
\text{COST}(\{C_j^*\}) = \sum_j w_j C_j^* \leq \text{COST}(\sigma^\lambda) + \sum_j w_j.
\]
From Lemma 4.1, Lemma 4.2, and Eq. (6), we have $\mathbb{E}[\text{Cost}\{\{C^*_j\}\}] \leq 2 \text{Cost}(\text{LP}_{\text{primal}})$. Thus, we have found integer deadlines, $\{C^*_j\}$, such that there exists a continuous schedule that completes each job $j$ by time $C^*_j$ and $\sum_j w_j C^*_j$ is at most twice the optimal LP$_{\text{primal}}$ cost in expectation.

We note that the random choice of $\lambda$ can be derandomized. To keep the flow of our presentation, we defer the derandomization to Section 6.

### 4.2 Finding a Feasible Integral Schedule Meeting Deadlines

This section is devoted to finding an integral schedule that complete all jobs before their respective deadlines, $\{C^*_j\}$. Recall that $\sigma^\lambda$ is such a schedule, except that it is continuous, not integral. We first convert the continuous time schedule $\sigma^\lambda$ back into a time-slotted fractional schedule such that each co-flow $j$ is completed by time $C^*_j$. Define $z_{i,o}^j = \int_{t=\tau}^{T_j} \sigma^\lambda_{i,o,j}(\tau)d\tau$ for all $t \leq \lceil \tau^\lambda_{i,o,j} \rceil$. It is easy to verify that the set $\{z_{i,o}^j\}$ of values define a feasible fractional co-flow schedule such that each coflow finishes by time $C^*_j$. In particular, we have the following.

\[
\sum_{t \leq C^*_j} \sum_{t \leq C^*_j} z_{i,o}^j = \sum_{t \leq C^*_j} \int_{t=\tau}^{T_j} \sigma^\lambda_{i,o,j}(\tau)d\tau = \int_{\tau=0}^{T_j} \sigma^\lambda_{i,o,j}(\tau)d\tau = d_{i,o}^j, \quad \forall j \in J \text{ and } \forall i, o \in [m] \quad (7)
\]

\[
\sum_{j \in J} \sum_{i,o \in [m]} z_{i,o}^j = \sum_{j \in J} \sum_{i,o \in [m]} \int_{t=\tau}^{T_j} \sigma^\lambda_{i,o,j}(\tau)d\tau \leq 1, \quad \forall t \in [T] \text{ and } \forall i \in [m] \quad (8)
\]

\[
\sum_{j \in J} \sum_{i \in [m]} z_{i,o}^j = \sum_{j \in J} \sum_{i \in [m]} \int_{t=\tau}^{T_j} \sigma^\lambda_{i,o,j}(\tau)d\tau \leq 1, \quad \forall t \in [T] \text{ and } \forall i \in [m] \quad (9)
\]

The last inequality in the above two statements follows from construction as $\sigma^\lambda$ schedules an integral matching at any instantaneous time $\tau$.

As a final step, we construct an instance of the network flow problem with integer capacities such that integral, maximum flows in this network are in one-to-one correspondence with integral feasible co-flow schedules. Our final integral co-flow schedule is obtained by solving a maximum flow problem on this network. This reduction to network flow is similar to the one used for the separation oracle in Section 3.3 and we highlight the differences below.

Construct a directed graph $G$ as follows. For each $t \in [T]$, let $G_t = (I_t, O_t, A_t)$ be a complete, bipartite, directed subgraph with unit capacity edges and vertices. For each job $j$ and pair $i, o \in [m]$, add vertex $source_{i,o,j}$ and connect it to all vertices $i_j$ for all $t \in [C^*_j] \setminus \tau_{r_j}$. Similarly, add vertex $sink_{i,o,j}$ and connect it to all vertices $o_t$ in the same range. Finally, add two new vertices $source$ and $sink$. For every coflow $j$ and pair $i, o \in [m]$, add an arc of capacity $d_{i,o}^j$ from $source$ to $source_{i,o,j}$ and also from $sink_{i,o,j}$ to $sink$. In contrast with the construction for the separation oracle that only considered a fixed co-flow $j$, here we add source and sink vertices for all co-flows.

Note that any integral maximum flow $f$ in $G$ of value $\sum_{j \in J} \sum_{i,o \in [m]} d_{i,o}^j$ corresponds to a feasible schedule $F$ such that any coflow $j$ has completion time at most $C^*_j$ and vice versa. In particular, for any coflow $j$ and pair $i, o \in [m]$, the maximum flow sends $d_{i,o}^j$ units of flow from $source_{i,o,j}$ to $sink_{i,o,j}$ through $d_{i,o}^j$ edge disjoint paths. The vertex capacities ensure that for any $t \in [T]$, the edges of $G_t$ carrying flow form a matching.

Finally we demonstrate that the $z_{i,o}^j$ values defined above yield a fractional flow $f$ of value $\sum_{j \in J} \sum_{i,o \in [m]} d_{i,o}^j$. Since all capacities in this network are integers, this guarantees the existence of an integral flow of the same value that can be found using an off-the-shelf network flow algorithm. Let $f(e)$
which is associated with a weight of job \( j \). For notational convenience, we don’t change the notation, so that \( I \) recursively split an interval if the interval contains the release time of some job \( j \) satisfy:

\[
1 + \frac{\text{optimal LP objective}}{\text{value of } I} \leq 1 + \epsilon
\]

and so on, where \( \kappa = O((1/\epsilon) \log T) \). Further, we recursively split an interval if the interval contains the release time of some job \( j \) until every interval has at most one job’s release time, and if so, it is the smallest time in the interval. Let \( \mathcal{I} \) denote the collection of the created intervals. For notational convenience, we don’t change the notation, so that \( I_1, I_2, \ldots, I_\kappa \in \mathcal{I} \) are ordered in increasing order of the times in them.

To compactly describe configurations, we do not distinguish times in same interval \( I_k \). Intuitively, as any two times in the same interval differ by a factor of at most \( 1 + \epsilon \), this compact description will increase the optimal LP objective by a factor of at most \( 1 + \epsilon \). For a job \( j \), a configuration is a complete, feasible, integral sketch for \( j \). Formally, a configuration \( F \) for job \( j \) is a ternary relation of tuples of the form \( (i, o, I) \), which is associated with a weight \( f(i, o, I) \), that indicates \( f(i, o, I) \) units of data are transferred from input port \( i \) to output port \( o \) during interval \( I \in \mathcal{I} \). For a feasible configuration \( F \) for job \( j \), the relation must satisfy:

1. \( \sum_{(i, o, I) \in F} f(i, o, I) = d_{io}^j \) for all \( i, o \in [m] \).
2. \( \sum_{(i, o, I) \in F} f(i, o, I) \leq |I| \) for every pair of \( i \in [m] \) and \( I \in \mathcal{I} \).
3. \( \sum_{(i, o, I) \in F} f(i, o, I) \leq |I| \) for every pair of \( o \in [m] \) and \( I \in \mathcal{I} \).

Let \( F(j) \) denote the set of all possible feasible configurations of \( j \). Let \( C^F_j \) denote the completion time of job \( j \) under configuration \( F \in F(j) \). More precisely, \( C^F_j := \max\{t \mid \exists i, o \in [m], I \in \mathcal{I} \text{ s.t. } f(i, o, I) > 0, t \in I\} \). In a similar spirit, define \( C(I) := \max\{t \in I\} \). We only need to change two constraints (2) and (3) of \( \text{LP}_{\text{primal}} \) as follows.
1. ∀i ∈ [m] and ∀I ∈ I, \( \sum_{j \in J, o \in [m]} \sum_{F \in F(j \times t)} y_j^F \cdot f(i, o, I) \leq |I| \).

2. ∀o ∈ [m] and ∀I ∈ I, \( \sum_{j \in J, i \in [m]} \sum_{F \in F(j \times t)} y_j^F \cdot f(i, o, I) \leq |I| \).

This new LP\(_{primal}^\prime\)’s optimal objective is at most \((1 + \epsilon)\) off the optimum, since times in the same interval differ by a factor of at most \((1 + \epsilon)\) and if a job completes during an interval, we pretend that it does at the last time in the interval. Note that since \(I\) has a polynomial number of intervals, the new LP\(_{primal}^\prime\) only has a polynomial number of constraints.

### 5.2 Solving the LP

The dual constraint (5) of LP\(_{dual}\) becomes:

\[ \forall j \in J \text{ and } \forall F \in F(j), \quad \alpha_j - \sum_{(i, o, I) \in F} (\beta_{i,t} + \gamma_{o,t}) \cdot f(i, o, I) \leq w_j C_j^F \]

Fix a job \(j\) and \(I_k \in I\). The key problem for the separation oracle becomes the following. During interval \(I\), input port \(i\) and output port \(o\) are priced at \(\beta_{i,t}\) and \(\gamma_{o,t}\), respectively, per unit traffic. Our task is to find a minimum-cost configuration for \(J\) that only uses \(I_1, I_2, \ldots, I_k \in I\). As before, we reduce the problem to an instance of the minimum cost flow problem. For each interval \(I \in \{I_1, I_2, \ldots, I_k\}\), instead of each \(t\), we create a complete, bipartite, directed subgraph \(G_I = (I_I, O_t, A_t)\). The cost of edge \((i_I, o_I)\) is set to \(\beta_{i,I} + \gamma_{o,I}\). All unit capacities in \(G_I\) are increased to \(|I|\). Similarly, source \(i_0\) is connected to vertices \(i_I\) for every interval \(I \in \{I_1, I_2, \ldots, I_k\}\) that does not include any time smaller than \(r_j\), with capacity \(|I|\). Likewise, the capacities of the edges connecting vertices \(o_I\) to sink \(o_0\) are increased to \(|I|\). Note that any integral maximum flow \(f\) in \(G\) corresponds to a feasible configuration \(F\) for job \(j\) with completion time at most \(C(I_k)\) and vice versa. Further, the cost of the flow sending \(f'(i, o, I)\) units of flow from \(i\) to \(o\) during interval \(I\) is exactly \(\sum_{i,o}(\beta_{i,t} + \gamma_{o,t}) \cdot f(i, o, I)\). Thus, we can obtain a feasible configuration \(F \in F(j)\) with completion time at most \(C(I_k)\) that minimizes \(\sum_{(i,o,t) \in F} (\beta_{i,t} + \gamma_{o,t}) \cdot f(i, o, I)\) from a minimum cost maximum flow in \(G\).

### 5.3 First Rounding

In this section, we adapt the rounding in Section 4.1 to avoid using the simplifying assumption. Since the entire rounding is almost identical, we only describe the key differences. For each \(I = \{t_1, t_2, \ldots, t_h\} \in I\), from an optimal solution to LP\(_{primal}\), we construct a continuous schedule that schedules the same integral matchings uniformly over the interval \([t_1, t_h]\). Then, we can still show \(\text{COST}(\sigma) \leq \text{COST}(\text{LP}_{\text{primal}}) - (1/2) \sum_j w_j - \epsilon\) —we can get a tighter bound but this bound suffices for our goal. The remaining procedure is exactly the same, so we can show \(\mathbb{E}[\sum_j w_j C_j^*] \leq 2 \text{COST}(\text{LP}_{\text{primal}})\).

### 5.4 Second Rounding

The second rounding also remains almost identical. There are two main differences that are worth special attention. First, we need to refine \(I\) to include the discovered deadlines \(C_j^*\). This is because we treat all times in the same interval in \(I\) equally, and as a result, we can’t strictly enforce jobs’ deadlines which could potentially yield an integral schedule whose cost is more than twice the optimal LP cost. Therefore, we recursively refine intervals, so that each interval \(I \in I\) contains at most one distinct deadline \(C_j^*\), and if so, it is the largest time in the interval. As before, we do not distinguish times in the same interval though. Thus, we will use \(I\) in place of \(t\) and increase the capacity of any edge involving interval \(I\) from one to \(|I|\).

The other key difference is this. Take a close look at the discovered integral maximum flow for a fixed interval \(I \in I\). Observe that in the flow, each node supports at most \(|I|\) units of flow. In other words, unit sized packets are delivered from input ports to output ports to (partially) serve some jobs and each port is
used by at most $|I|$ packets. To obtain a feasible integral schedule, we convert this into $|I|$ integral matchings. As mentioned earlier, it was already observed in [10] that such a conversion is possible: a bipartite graph can be decomposed into a set of integral matchings whose number is equal to the graph’s maximum vertex degree. Thus, we can find in polynomial time an integral schedule that is 2-approximate against the optimal LP$_{\text{primal}}$ cost. This implies the discovered schedule is $2(1+\epsilon)$-approximation, as the relaxation is at most $(1+\epsilon)$ factor off the optimum.

6 Derandomization

In this section, we discuss how to derandomize the random choice of $\lambda \in (0,1]$, which was used in the first rounding to construct a stretched schedule $\sigma^\lambda$ from $\sigma$. Let us first define step values. We say that $v \in (0,1]$ is a step value if $\sum_{F \in \mathcal{F}(j); C_F^j \leq C} y^F_j = v$ for some $j \in J$ and integer $C \geq 1$.— in other words, exactly $v$ fraction of some job $j$’s configurations are completed by some integer time in the LP$_{\text{primal}}$ solution, $\{y^F_j\}$. Let $V$ denote the set of all step values; $1 \in V$ by definition. We can assume w.l.o.g. that $|V|$ is polynomially bounded, as we can find an optimal solution to LP$_{\text{primal}}$ with a polynomial-size support.

We have shown in Section 4.1 that $E[\text{cost}(\sigma^\lambda)] \leq 2\text{cost}(\text{LP}_{\text{primal}}) - \sum_j w_j$ and $\text{cost}(\{C^*_j\}) \leq \text{cost}(\sigma^\lambda) + \sum_j w_j$. Therefore, if we find $\lambda$ such that $\text{cost}(\sigma^\lambda) \leq 2\text{cost}(\text{LP}_{\text{primal}}) - \sum_j w_j$ (it is easy to see such a $\lambda$ exists from a simple averaging argument), then we can find $\{C^*_j\}$ such that $\text{cost}(\{C^*_j\}) \leq 2\text{cost}(\text{LP}_{\text{primal}})$.

Our remaining goal is to find $\lambda$ that minimizes $\text{cost}(\sigma^\lambda)$. Towards this end, fix a job $j$ and take a close look at $j$’s contribution to $\text{cost}(\sigma^\lambda)$ depending on the choice of $\lambda$. Fix any two adjacent step values $v_1 < v_2$ in $V$. Suppose $\lambda$ was set to a value $v \in (v_1, v_2]$. Let $t$ be the earliest time slot such that $\sum_{F \in \mathcal{F}(j); C_F^j \leq t} y^F_j \geq v_2$. Let $v_0 := \sum_{F \in \mathcal{F}(j); C_F^j \leq t} y^F_j$; note that $v_0 \leq v_1$, as $v_1$ and $v_2$ are adjacent step values. Then, we have $C_{\sigma^\lambda}(j) = \frac{1}{v}(t - 1 + \frac{v_1 - v_0}{v_2 - v_1})$. This is because exactly $v_0$ fraction of $j$’s configurations complete by time $t - 1$ and there is $(v_2 - v_0)$ fraction of $j$’s configurations completing at time $t$; thus, $C_{\sigma^\lambda}(j) = t - 1 + \frac{v_1 - v_0}{v_2 - v_1}$. Hence, for any $v \in (v_1, v_2]$, $j$ contributes to $\text{cost}(\sigma^\lambda)$ by $w_j \frac{1}{v}(t - 1 + \frac{v_1 - v_0}{v_2 - v_1})$. This becomes a linear function in $z$ over $[1/v_2, 1/v]$ if we set $z = 1/v$. Therefore, we get a piece-wise linear function $g(z)$ by summing over all jobs and considering all pairs of two adjacent step values in $V$. We set $\lambda$ to the the inverse of $z$’s value that achieves the global minimum, which can be found in polynomial time.

7 Extensions To Non-Uniform Capacity Constraints

In this section, we consider the co-flow scheduling problem with non-uniform port capacities. The problem setting remains exactly the same as regular co-flow scheduling, with the exception that port $i$ has an integer capacity $c_i$ that denotes the number of packets that can be processed by port $i$ at any time $t$. We show that our algorithm can be easily extended to give the same approximation guarantee even for this generalized problem. Because the key ideas remain the same, we only highlight the primary differences in each step.

7.1 LP Formulation and Separation Oracle

Let $\mathcal{F}(j)$ denote the set of all feasible configurations of job $j$. Note that a configuration $F \in \mathcal{F}(j)$ may schedule up to $c_i$ units of data transfer from (or to) port $i$ at time slot $t$. For an input port $i$, let $\text{load}_F(i,t)$ denote the number of packets transferred from port $i$ at time $t$ as per configuration $F$. We similarly define $\text{load}_F(o,t)$ for an output port $o$. We only need to change constraints (2) and (3) as follows.

$$\forall i \in [m] \text{ and } \forall t \in [T], \sum_{j \in J} \sum_{F \in \mathcal{F}(j)} \text{load}_F(i,t) y^F_j \leq c_i$$

$$\forall o \in [m] \text{ and } \forall t \in [T], \sum_{j \in J} \sum_{F \in \mathcal{F}(j)} \text{load}_F(o,t) y^F_j \leq c_o$$
The separation oracle for the dual LP once again relies on finding the minimum cost schedule for a job \( j \) subject to a fixed deadline. Similar to Section 3.3, this key problem can be solved using minimum cost flow. Since the reduction is almost identical to the one earlier, we skip the details.

### 7.2 Rounding

The key difference in the rounding step occurs in the construction of the continuous schedule \( \sigma \). The fractional schedule obtained from the LP relaxation no longer schedules a fractional matching during a time slot \( t \). Let \( X^t \) be the bipartite graph scheduled by the feasible LP solution at time slot \( t \). We are guaranteed that the fractional degree of each vertex \( i \in X^t \) is at most \( c_i \). To construct the continuous schedule, we need to decompose \( X^t \) into a convex combination of a polynomial number of degree bounded subgraphs \( B_1, B_2, \ldots, B_\ell \). Such a decomposition can be obtained by splitting a vertex \( i \in X^t \) into \( c_i \) distinct copies and splitting the edge weights so that each vertex has at most unit fractional degree in the expanded graph. The Birkhoff-von Neumann decomposition of the expanded graph then gives us the desired decomposition.

Once we have the continuous time schedule \( \sigma \), the rest of the rounding procedure remains identical to that in Sections 4.1 and Sections 4.2.

### References

[1] Saba Ahmadi, Samir Khuller, Manish Purohit, and Sheng Yang. On scheduling coflows. In IPCO, pages 13–24. Springer, 2017.

[2] Nikhil Bansal and Subhash Khot. Inapproximability of hypergraph vertex cover and applications to scheduling problems. In ICALP, pages 250–261. Springer, 2010.

[3] Zhi-Long Chen and Nicholas G Hall. Supply chain scheduling: Conflict and cooperation in assembly systems. Operations Research, 55(6):1072–1089, 2007.

[4] Mosharaf Chowdhury and Ion Stoica. Coflow: A networking abstraction for cluster applications. In ACM Workshop on Hot Topics in Networks, pages 31–36. ACM, 2012.

[5] Mosharaf Chowdhury and Ion Stoica. Efficient coflow scheduling without prior knowledge. In SIGCOMM, pages 393–406. ACM, 2015.

[6] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with varys. In SIGCOMM, SIGCOMM ’14, pages 443–454, New York, NY, USA, 2014. ACM.

[7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. Communications of the ACM, 51(1):107–113, 2008.

[8] Naveen Garg, Amit Kumar, and Vinayaka Pandit. Order scheduling models: Hardness and algorithms. In FSTTCS, pages 96–107. Springer, 2007.

[9] Sungjin Im, Maxim Sviridenko, and Ruben Van Der Zwaan. Preemptive and non-preemptive generalized min sum set cover. Mathematical Programming, 145(1-2):377–401, 2014.

[10] Hamidreza Jahanjou, Erez Kantor, and Rajmohan Rajaraman. Asymptotically optimal approximation algorithms for coflow scheduling. CoRR, abs/1606.06183, 2016.

[11] Samir Khuller, Jingling Li, Pascal Sturmfels, Kevin Sun, and Prayaag Venkat. Select and permute: An improved online framework for scheduling to minimize weighted completion time. arXiv preprint arXiv:1704.06677, 2017.
[12] Samir Khuller and Manish Purohit. Brief announcement: Improved approximation algorithms for scheduling co-flows. In Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, pages 239–240. ACM, 2016.

[13] Joseph Y-T Leung, Haibing Li, and Michael Pinedo. Scheduling orders for multiple product types to minimize total weighted completion time. Discrete Applied Mathematics, 155(8):945–970, 2007.

[14] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li. Towards practical and near-optimal coflow scheduling for data center networks. IEEE Transactions on Parallel and Distributed Systems, PP(99):1–1, 2016.

[15] Monaldo Mastrolilli, Maurice Queyranne, Andreas S Schulz, Ola Svensson, and Nelson A Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. Operations Research Letters, 38(5):390–395, 2010.

[16] Zhen Qiu, Cliff Stein, and Yuan Zhong. Minimizing the total weighted completion time of coflows in datacenter networks. In SPAA, SPAA ’15, pages 294–303, New York, NY, USA, 2015. ACM.

[17] Maurice Queyranne and Maxim Sviridenko. A (2+ ε)-approximation algorithm for the generalized preemptive open shop problem with minsum objective. Journal of Algorithms, 45(2):202–212, 2002.

[18] Sushant Sachdeva and Rishi Saket. Optimal inapproximability for scheduling problems via structural hardness for hypergraph vertex cover. In IEEE Conference on Computational Complexity, pages 219–229. IEEE, 2013.

[19] Andreas S Schulz and Martin Skutella. Random-based scheduling new approximations and lp lower bounds. In International Workshop on Randomization and Approximation Techniques in Computer Science, pages 119–133. Springer, 1997.

[20] Mehrnoosh Shafiee and Javad Ghaderi. An improved bound for minimizing the total weighted completion time of coflows in datacenters. arXiv preprint arXiv:1704.08357, 2017.

[21] Guoqing Wang and TC Edwin Cheng. Customer order scheduling to minimize total weighted completion time. Omega, 35(5):623–626, 2007.

[22] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. HotCloud, 10(10-10):95, 2010.

[23] Yangming Zhao, Kai Chen, Wei Bai, Minlan Yu, Chen Tian, Yanhui Geng, Yiming Zhang, Dan Li, and Sheng Wang. Rapier: Integrating routing and scheduling for coflow-aware data center networks. In INFOCOM, pages 424–432. IEEE, 2015.