CS 2420 - Algorithm Design, Analysis and Theory

Lecture #3: 29 September 2010

Lecturer: Brendan Lucier

Scribe Notes by: Anastasios Zouzias

1 Outline of this Lecture

- 2-approximation for 2-restricted makespan problem
- 2-approximation for unrelated machines
- Review on graph flow-based algorithms

2 2-approximation for 2-restricted Makespan Problem

Assume the scheduling problem on m machines with n jobs, where each job j has different processing time on every machine, denote the processing times by a m-tuple $(p_{j_1}, p_{j_2}, \ldots, p_{j_m})$,where p_{j_l} is the processing time of job j on machine l. In the restricted model, we assume that $p_{j_l} \in \{p_j, \infty\}$, i.e., each job can be scheduled only on some subset of machines and has the same processing times. In this section we will focus on the 2-restricted model, where we also assume that for every job j

$$|\{i \mid p_{j_i} < \infty, i \in [m]\}| \leq 2.$$

To solve this scheduling problem we will reduce it to the following graph problem. Let $G = (V, E, w)$ be a *undirected* weighted graph (allowing self-loops and multi-edges) with positive weights $w_e \geq 0$ for all $e \in E$. The goal is to *orient* each edge of G so that the maximum over all the vertices of the sum of incoming edge weights is minimized. We call this the *graph orientation problem*.

The Reduction

Starting with a 2-restricted makespan instance we will constuct an instance of the graph orientation problem. For each machine, construct a new vertex v. For each job $j \in [n]$ that can scheduled to exactly two machines, construct an (un-directed) edge with weight p_j between the vertices that correspond to these two machines. Otherwise if a job can be scheduled to only one machine, then add a self-loop on the corresponding vertex.

It is easy to see that, by construction, an orientation on the above graph that minimizes the maximum over all the vertices of the sum of incoming edge weights corresponds to an optimum schedule for the 2-restricted makespan problem. Next we describe a 2-approximation algorithm for the graph orientation problem.

The Algorithm

We will formulate the graph orientation problem as a integer program.

$$\begin{aligned} \min. \quad & T \\ \text{subject to} \quad & x_{eu} + x_{ev} = 1, \quad \forall e \in E \\ & q_v + \sum_{e:\ v \in e} x_{ev} \cdot p_e \leq T, \quad \forall v \in V \\ & x_{ev} \in \{0,1\}, \quad \forall v \in V \end{aligned}$$

where q_v is the sum of processing times of jobs that can only assigned to the machine that correspond to the vertex v. Roughly speaking, the intuition is that if the variable $x_{ev} = 1$, this means that we orient the edge $e = \{u, v\}$ in direction of v. T is also the "objective value" of the graph orientation problem. Now we relax the above program to a LP, by letting $x_{ev} \in [0, 1]$. As in the previous lecture, we can assume that we know the optimum value $T = T^*$ of the graph orientation problem, by using binary search. Given an instance of the graph orientation problem we run the corresponding LP. Assume that the linear program returns a solution x^*_{ev} for all $e \in E, v \in V$, say with makespan T. In general such a solution is fractional, hence we would like to find an integral solution x^{int}_{ev} with makespan at most $2T$. First set $x_{ev} = 1$ whenever $x^*_{ev} = 1$. Consider the graph H of fractionally oriented edges. The rounding process consists of two steps:

(i) Adjust the fractional solution until H has no cycles while maintaining the objective value of the fractional solution to be at most T.

(ii) Round the fractional solution of the produced acyclic graph to an integral solution by increasing the objective value by an additive factor T.

Step 1: Now consider a cycle in the undirected graph H. The main observation is that one edge in C is now not fractional, so we can remove it from H. Also, the new fractional solution is non-negative and more importantly we haven't increased the (fractional) cost of our fractional solution. By iterating this process, we can eliminate all fractional cycles.

Step 2: We know that H has no cycles. Hence H is a forest. For each connected component, which is a tree, orient the edges towards the leaves of the trees. The main observation is that the in-degree of each vertex is increased by at most one. So, in particular each machine gets an overhead of at most T in worst case. This observation combined with Step 1, implies that we get a bound of $2T$ on the makespan.

Algorithm 1 Cycle Elimination

procedure CYCLE ELIMINATION(Directed graph $H = (V, E, x_{ev})$, cycle C in H)
  Orient the cycle arbitrarily (clock-wise or counter-clock-wise).
  Let $\delta \leftarrow \min_{e \in C} x^*_{ev} p_e$ (minimum over all $e : v \to u$ that agree with the above orientation of C) and e_δ the edge that minimizes the above.
  Set $x^*_{eu} \leftarrow x^*_{ev} - \delta / p_{e_\delta}$ for every edge that has the same orientation with Step 2.
  Set $x^*_{ev} \leftarrow x^*_{ev} + \delta / p_{e_\delta}$ for every edge in opposite direction w.r.t. the orientation of Step 2.
  Output: The graph H with C cycle eliminated.
end procedure

3 2-approximation for Unrelated Machines

$$\begin{aligned} & \text{Find feasible solution} \\ \text{subject to} \quad & \sum_{i \in [m]} x_{ji} = 1, \quad \forall e \in E \\ & \sum_{i \in [m]} x_{ji} p_{ji} \le T^*, \quad \forall i \in [m] \\ & x_{ji} \in \{0, 1\}, \quad \forall (j, i) \in [n] \times [m] \end{aligned}$$

Again, the approach here is to relax the above integer program to an LP by allowing $x_{ji} \in [0, 1]$. Then solve the LP to get a fractional solution x^*_{ji}, and finally round the fractional solution to find an integral solution with makespan at most $2T$.

We will give a sketch of the proof. We use the following two results that exploit the structure of the above specific LP.

Lemma 1. *If there exists a solution to the LP, then there exists a solution with at most $m + n$ non-zero variables.*

Also we can give a bound on the number of fractionally assignment jobs.

Theorem 2. *There are at most m fractionally assigned jobs.*

The main idea of the proof is as follows: Construct a bipartite graph of size $m + n$ where the left vertices correspond to the fractionally assigned jobs, and the right side of vertices correspond to the machines. We put an edge between a fractionally assigned job to a machine with fractional weight. Then we can prove the following claim.

Claim 3. *There is perfect matching between the fractionally assigned jobs and the machines.*

Hence we can round according to this perfect matching. So, each machine gets at most one extra job, and therefore the total makespan is at most $2T$.

4 Review on graph flow-based algorithms

We start by defining a *flow network*. A flow network $\mathcal{F} = (G, E, s, t)$ consists of a bidirected graph $G = (V, E)$, a capacity function $c : E \to \mathbb{R}_{\ge 0}$, and two vertices of G, s and t that are the *source*

and *terminal* of the flow network. A *flow* is a function $f : E \to \mathbb{R}$ which satisfies that following properties:

(i) $f(e) \leq c(e)$, $\forall e \in E$. This means that we can't "transmit" more flow than the capacity of each edge.

(ii) $f(u, v) = -f(v, u)$, $\forall e = (u, v)$, i.e., the flow is preserved over each edge.

(iii) $\forall u \in V \setminus \{s, t\}$, $\sum_{f(u,v)} = 0$. This is similar with Kirchhoff's circuit law for electrical flows, i.e., the amount of incoming flow equals the amount of outcoming flow for every vertex.

Given a flow network $\mathcal{F}$, we define the value of the flow as $\text{val}(\mathcal{F}) = \sum_{v \in N(s)} f(s, v)$, where $N(s) = \{v \mid (s, v) \in E\}$. Our goal is to find a maximum flow on a flow network. To continue our discussion we need the following list of definitions.

Residual Graph: Given a flow f, we define the residual graph G_f on the same vertex set with the flow network as $E(G_f) = \{e \mid c_f(e) > 0,\ e \in E\}$ where $c_f(e) = c(e) - f(e)$.

Residual Network: The residual network is defined as $\mathcal{F} = (G_f, c_f, t, s)$.

Augmenting Path: A path Π from s to t in G_f is called an augmenting path. We will also denote by $c_f(\Pi) = \min_{e \in \Pi} c_f(e)$, the "left-over" capacity of the path Π.

Augmenting flow An augmenting flow f_Π with respect to a augmenting path Π is defined as follows

$$f_\Pi(e) := \begin{cases} c_f(\Pi) & \text{if } e \in \Pi, \\ 0 & \text{otherwise.} \end{cases}$$

The following greedy algorithm finds a maximum flow of a given flow network by adding to the current flow greedily an augmenting flow. The following theorem proves the correctness of the

Algorithm 2 Ford-Fulkerson

```
1: procedure MAX-FLOW(A flow network F = (G, E, s, t))
2:     Initialize f(e) = 0 for all e ∈ E.
3:     while There exists an augmenting path Π in G_f do
4:         Update f ← f + f_Π
5:     end while
6:
7:     Output: A maximum flow f : E → ℝ.
8: end procedure
```

above algorithm (proof omitted).

Theorem 4. *The following are equivalent:*

(a) f is a maximum flow.

(b) There is no augmenting path for f.

(c) There is a cut $s - t$ with value exactly val(f).

Remark 1. *If the capaticites of the flow network are integral, then there exists a maximum flow with integral value.*