15-854: Approximations Algorithms **Lecturer:** R. Ravi
Topic: Greedy Algorithms : Minimizing Makespan, Multiway Cut **Date:** 9/19/2005
Scribe: Vineet Goyal

3.1 Introduction

In this lecture, we discuss a simple and generic paradigm of greedy algorithms. A nice thing about it is that once the problem has been stated, the greedy paradigm naturally translates into a simple algorithm. We introduce it with the greedy algorithms for minimum makespan scheduling and multiway cut problems in this lecture.

3.2 Minimum Makespan Scheduling

A central problem in scheduling theory is to design a schedule such that the last finishing time of the given jobs (also called *makespan*) is minimized. This problem is called the *minimum makespan scheduling.*

Problem Definition Given processing times for n jobs, $p_1, p_2, \ldots, p_n$, and an integer m, find an assignment of the jobs to m identical machines so that the completion time, also called the *makespan*, is minimized.

Minimum makespan scheduling is NP-hard as we can reduce 2-PARTITION to minimum makespan scheduling on two machines. Note that this reduction only proves that minimum makespan scheduling is weakly NP-hard. This infact is the case when there are a constant number of machines. Horowitz and Sahni [7] give an FPTAS for this variant of the problem. However, the minimum makespan problem is strongly NP-hard for m arbitrary (by a reduction from 3-PARTITION [3]). Thus, there cannot exist an FPTAS for the minimum makespan problem in general, unless P=NP. However, Hochbaum and Shmoys [6] gave a PTAS ($(1+\epsilon)$-approximation algorithm which runs in polynomial time for any fixed ϵ) for the problem.

We can consider generalization of the minimum makespan scheduling, where the machines are not identical and a job has different processing times on different machines. This version is called as Scheduling on Unrelated Parallel Machines. Lenstra, Shmoys, and Tardos [9] gave a 2-approximation for this version. They also proved that it is not possible to approximate it within a factor $(3/2-\epsilon)$ for any $\epsilon > 0$, unless P=NP. Other generalizations include, considering precedence constraints in the scheduling of jobs. Lam and Sethi [8] gave a $(2-2/n)$-approximation for the version where each job has a unit processing time.

In this lecture, we focus on the minimum makespan scheduling, where all the machines are identical and show that a greedy algorithm gives a $(2-1/m)$-approximation. The results we present are due to Graham [4]. Note that the completion time for a machine only depends on the set of jobs scheduled on it irrespective of the order of processing.

The greedy algorithm is as follows :

Algorithm Minimum Makespan Scheduling

1. Order the jobs arbitrarily.
2. Schedule jobs on machines in this order, scheduling the next job on the machine that has been assigned the least amount of work so far.

This algorithm is also called *list scheduling* algorithm since we can view it as assigning a list of numbers (job sizes) to one of the m bins in a greedy fashion. We prove that the above mentioned greedy algorithm gives a 2-approximation.

To analyse the performance of this algorithm, we need to come up with a lower bound on the makespan of any optimal schedule. The general technique for constructing a lowerbound is to inspect the structure of any optimal solution and derive a lowerbound from that.

Fix an optimal schedule with makespan OPT. Consider the schedule returned by the above greedy algorithm and suppose the earliest finishing time of any machine for that schedule is t_F. We claim that t_F is at most the average load on the machines.

Fact 3.2.1 $t_F \leq \frac{\sum_{i=1}^{n} p_i}{m}$

Otherwise, the total load on all the machines will be strictly greater than $\sum_{i=1}^{n} p_i$, a contradiction.

Fact 3.2.2 $OPT \geq \frac{\sum_{i=1}^{n} p_i}{m}$

Otherwise, the total load processed by OPT will be strictly less than $\sum_{i=1}^{n} p_i$, a contradiction. Fact 3.2.1 and 3.2.2 imply $\text{OPT} \geq t_F$.

Another trivial lowerbound on OPT is the maximum processing time of a job. Let $p_{max} = \max_i p_i$, then $\text{OPT} \geq p_{max}$.

Theorem 3.2.3 *Algorithm Minimum Makespan Scheduling is a $(2 - 1/m)$-approximation.*

Proof: Let machine j have the maximum completion time (say t_{max}) in the schedule returned by the greedy algorithm. Let p_l be the last job assigned to j. Consider the time, (say t_l), when p_l starts processing on machine j. Clearly, at time t_l all other machines are busy as otherwise p_l would have been scheduled earlier on a different machine by the greedy algorithm. Thus, by an averaging argument we have,

$$t_l + \frac{p_l}{m} \leq \text{OPT}$$

which implies $t_l \leq \text{OPT} \cdot (1 - 1/m)$ as $\text{OPT} \geq p_l$. Hence, the makespan, T of the greedy schedule is,

$$\begin{aligned} T &= t_l + p_l \\ &\leq (1 - 1/m) \cdot \text{OPT} + \text{OPT} \\ &= (2 - 1/m) \cdot \text{OPT} \end{aligned}$$

■

The above analysis is tight i.e. there exist instances where the greedy algorithm produces a schedule which has makespan $(2 - 1/m)$ times the optimal. Consider the following instance : $(m^2 - m)$ jobs with unit processing time and a single job with processing time m. Suppose the greedy algorithm schedules all the unit jobs before the long job, then the makespan of the schedule obtained is $(2m-1)$ while the optimal makespan is m. Hence, the algorithm gives a schedule which has makespan $(2-1/m)$ times the optimal.

Note that in the above instance if we schedule the longest job first then we obtain an optimal schedule. So, we might think that considering jobs in order of non-increasing processing times might give us a better approximation guarantee. Infact, we can prove the following theorem :

Theorem 3.2.4 *[5] The variant of Algorithm Minimum Makespan Scheduling, where we order jobs by decreasing processing time gives a* $4/3$*-approximation for the minimum makespan scheduling problem.*

It is easy to prove that the above variant gives a $3/2$-approximation and is one of the exercises in the homeworks. However, the best approximation for the minimum makespan problem is a polynomial time approximation scheme (PTAS) due to Hochbaum and Shmoys [6]. The minimum makespan problem is closely related to the bin packing problem : there exists a schedule of makespan T, if and only if n objects of sizes $p_1, p_2, \ldots, p_n$ can be packed into m bins of capacity T each. Let $\text{bins}(T)$ denote the minimum number of bins of size T required to pack $p_1, \ldots, p_n$. Thus, the minimum makespan problem can be formulated as

$$\min\{T : \text{bins}(T) \leq m\}$$

In [6], Hochbaum and Shmoys use the above observation to obtain a PTAS. The idea is the following: job sizes are scaled and rounded so that there are a bounded (constant) number of different sizes and then a dynamic programming algorithm is used to solve the resulting bin-packing problem.

3.3 Minimum Multiway Cut

In this section, we present a greedy algorithm for the minimum multiway cut problem. The minimum multiway cut problem is a generalization of the min-cut problem which is defined as : given an undirected graph $G = (V, E)$ with non-negative weights w on edges and terminals $s, t \in V$, the objective is to find a minimum weight subset of edges E', such that removing E' from G disconnects s from t. Minimum s-t cut has a nice combinatorial characterization from the max flow min cut theorem. Thus, min-cut problem can be solved exactly in polynomial time using a maximum flow algorithm.

Minimum multiway cut Given a set of terminals $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$, a multiway cut is a set of edges whose removal from G disconnects the terminals from each other. The objective is to find a minimum weight multiway cut.

The problem is NP-hard for $k \geq 3$ [2]. Note that for $k = 2$, it reduces to the min-cut problem which can be solved exactly. The greedy algorithm is described below.

Algorithm Multiway Cut

1. For each $i = 1, 2, \ldots, k$, compute a minimum weight s_i-$S \setminus \{s_i\}$ cut, say C_i.

2. Discard a cut with the maximum weight and output the union of rest, say C.

To compute a minimum weight s_i-$S \setminus \{s_i\}$ cut, identify the terminals in $S \setminus \{s_i\}$ into a single node, say t_i and find the minimum weight s_i-t_i cut in the modified graph. Clearly, removing C from the graph G disconnects every pair of terminals, and so C is a multiway cut. The above algorithm requires k min-cut computations. To analyse the performance, we consider an optimal solution and try to relate it to the solution returned by our algorithm.

Theorem 3.3.1 *Algorithm Multiway Cut achieves an approximation guarantee of* $2 - 2/k$.

Proof: Let E^* be an optimal multiway cut. Removing E^* from G results in exactly k components. Let O_i be the connected component containing terminal s_i as shown in the Figure 3.3.1. For any subset $X \subseteq V$, $\delta(X)$ denotes the set of edges which have exactly one end point in X.

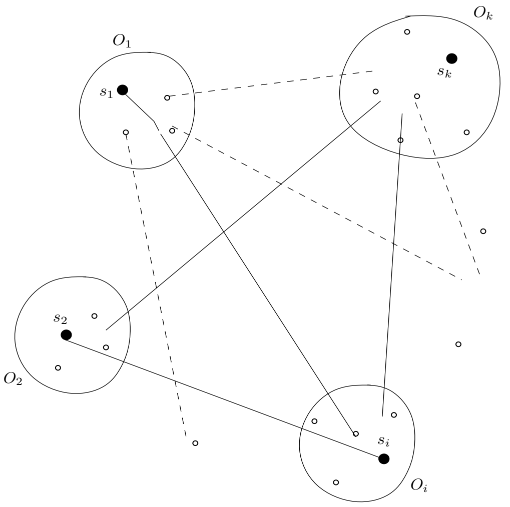

Figure 3.3.1: Components $O_1, O_2, \dots, O_k$ are formed after removing E^* from G. Edges between the components are the optimal edges.

Note that $\delta(O_i)$ is an s_i-$S \setminus \{s_i\}$ cut. Since C_i is a minimum weight s_i-$S \setminus \{s_i\}$ cut, we have

$$w(C_i) \leq w(\delta(O_i)).$$

Thus,

$$\sum_{i=1}^{k} w(C_i) \leq \sum_{i=1}^{k} w(\delta(O_i)) \leq 2 \cdot w(E^*)$$

since, each edge of E^* gets counted twice in the summation.

Now, the final solution, C of our greedy algorithm is obtained by discarding the maximum weight cut among $C_1, C_2, \dots, C_k$. The maximum weight cut has weight at least $\frac{1}{k} \sum_{i=1}^{k} w(C_i)$. Thus,

$$w(C) \leq (1 - \frac{1}{k}) \sum_{i=1}^{k} w(C_i) \leq 2(1 - \frac{1}{k}) w(E^*).$$

■

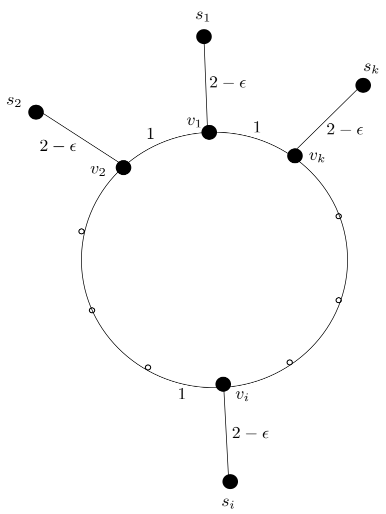

Figure 3.3.2: Tight example for the greedy algorithm for multiway cut. The set of terminals, $S = \{s_1, s_2, \dots, s_k\}$.

The analysis in the above proof is tight as shown by the example in Figure 3.3.2: Let $V = \{s_1, s_2, \dots, s_k, v_1, v_2, \dots, v_k\}$. There is a k-cycle, $(v_1, v_2, \dots, v_k, v_1)$ and s_i is adjacent to v_i for all $i = 1, 2, \dots, k$. All edges in the cycle have a unit weight and edge (v_i, s_i) has weight $2 - \epsilon$ for all $i = 1, \dots, k$. The set of terminals,$S = \{s_1, \dots, s_k\}$. For each i, minimum s_i-$S \setminus s_i$ cut is the edge (v_i, s_i). Thus, the greedy algorithm returns the edge set $\{(v_1, s_1), (v_2, s_2), \dots, (v_{k-1}, s_{k-1})\}$ which has weight $(k-1) \cdot (2 - \epsilon)$. However, the optimal solution is remove the cycle which has weight k. Thus, the greedy algorithm returns a solution which is $2(1 - 1/k)$ times the optimal (since ϵ can be arbitrary small).

The best known approximation algorithm for this problem is due to Calinescu et al. [1] and it achieves an approximation guarantee of $(3/2 - 1/k)$.

References

[1] Gruia Calinescu, Howard Karloff, and Yuval Rabani. An improved approximation algorithm for multiway cut. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 48–52, New York, NY, USA, 1998. ACM Press.

[2] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiway cuts (extended abstract). In *STOC '92: Proceedings of the twenty-*

fourth annual ACM symposium on Theory of computing, pages 241–251, New York, NY, USA, 1992. ACM Press.

[3] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[4] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[5] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

[6] Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17(3):539–551, 1988.

[7] Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *J. ACM*, 23(2):317–327, 1976.

[8] S. Lam and R. Sethi. Worst case analysis of two scheduling algorithms. *SIAM Journal on Computing*, 6(3):518–536, 1977.

[9] J. K. Lenstra, D. B. Shmoys, and Eva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46(3):259–271, 1990.