CSC 2421 Approximation Algorithms
Lecture 2, Spring 2005

Lecturer: Allan Borodin
Scribe: Michael Pavlin
Department of Computer Science
University of Toronto

1 Introduction

This lecture presents a number of examples of greedy algorithms for approximate solutions to scheduling problems. We present a syntactic definition of a greedy algorithm. Finally, we discuss an algorithm which does not fully correspond to our definition of a greedy algorithm but is intuitively greedy.

2 Definitions

An algorithm is a $c - approx$ algorithm for a maximization problem if for
$c \geq 1, \forall$ input $\mathcal{I}$, $w(OPT[\mathcal{I}]) \leq c \cdot w(ALG[\mathcal{I}])$
where $w(alg[\mathcal{I}])$ is the cost of the solution algorithm alg finds given input $\mathcal{I}$.

An alternative and more standard way of expressing this relationship for maximization problems is that an algorithm is an $\alpha - approx$ algorithm if for
$\alpha \leq 1, w(ALG[\mathcal{I}]) \geq \alpha \cdot w(OPT[\mathcal{I}])$

For minimization problems a $c - approx$ algorithm always has the following meaning: $\forall \mathcal{I}$
$w(ALG[\mathcal{I}]) \leq c \cdot w(OPT[\mathcal{I}])$

These notions can be extended to consider asymptotic behaviour by adding a constant d; i.e.
$\exists d, \forall \mathcal{I}\ w(OPT[\mathcal{I}]) \leq c \cdot w(ALG[\mathcal{I}]) + d$

3 Greedy Algorithms for Scheduling

As mentioned in the previous lecture, greedy algorithms are often the second approach we consider when trying to solve a problem. The first is a brute force search. Greedy algorithms often produce a good approximation and sometimes even optimal answers for natural problems. This applies to scheduling problems.

3.1 Scheduling Problems

Here are two classical classes of scheduling problems

1. Makespan problems where we are attempting to find a schedule which minimizes

the maximum completion time.

2. Job selection so as to maximize profit.

We focus on several instances of the second class of problems.

Interval Scheduling Problem Given the set of intervals
$\mathcal{I} = \{I_1, I_2, ..., I_n\}$
where
$I_j = (s_j, f_j)$ in the non-weighted version (ISP) and
$I_j = (s_j, f_j, w_j)$ in the weighted case (WISP).

Schedule intervals onto $m \geq 1$ machines so as to maximize profit subject to there being no intersection of intervals on a given machine. Profit is defined as the number of intervals scheduled in the ISP and the sum of the weights of the scheduled intervals in the WISP.

A Feasible Schedule obeys the following:
$\sigma : \{1, 2, ..., n\} \rightarrow \{0, 1, ..., m\}$
Subject to $\sigma(i) = \sigma(j) > 0, i \neq j \Rightarrow [s_i, f_i) \bigcap [s_j, f_j) = \emptyset$

Graph theoretic interpretation There is an equivalent graph theoretic interpretation which for $m = 1$ corresponds to finding the maximum weight independant set in an interval graph described as follows:
$\mathcal{I} = \{N_1, N_2, ..., N_n\}$ nodes which correspond to the original intervals
where
$N_i = \{N_1^i, ..., N_{d(i)}^i\}$ has adjacent nodes for each intersecting interval.

This problem is equivalent to ISP with respect to polynomial time. The conversion to the graph theoretic version can be expensive since there are up to n^2 edges.

A natural greedy algorithm for this problem is to choose intervals with smallest degree. This yields an approximation ratio of $3/2$.

Interval colouring problem The interval colouring problem (ICP) is closely related to ISP. We are presented with the same inputs $\mathcal{I} = I_1, ..., I_n$, but ICP attempts to find the minimum machine schedule where all intervals are scheduled. The following algorithm provides an optimal colouring.

ICP Greedy Algorithm:

Sort intervals such that $s_1 \leq s_2 \leq ... \leq s_n$
for $c = 1..n$ **do**
  use smallest "free colour" to colour I_c
end for

Greedy algorithms for the ISP With respect to ISP, the "Most Obvious" greedy algorithm is shortest processing time (SPT). The following is the single machine version of this algorithm:

SPT:

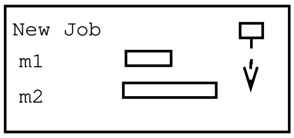
$p_i = f_i - s_i$
Sort intervals such that $p_1 \leq p_2 \leq ... \leq p_n$
$S = \emptyset$
for $j = 1..n$ **do**
  if I_j doesn't conflict with $\{I_k | k \in S\}$ **then**
    then $S = S \bigcup \{j\}$
  end if

end for

The optimal greedy ordering orders intervals by their finishing time.

Earliest Finish (EFT):
  Sort intervals such that $f_1 \leq f_2 \leq ... \leq f_n$
  $S = \emptyset$
  for $j = 1..n$ **do**
    if I_j doesn't conflict with $\{I_k | k \in S\}$ **then**
      then $S = S \bigcup j$
    end if
  end for

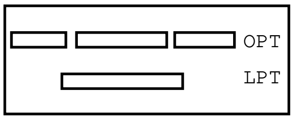
In the multiprocessor case we need to choose a machine: we simply put it on the machine which minimizes the gap between the new start time of the new interval and the finishing time of the preceeding interval.



For WISP we consider two variations on how the profit is calculated:

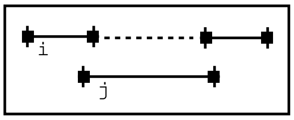
1. Proportional Profit $w_i = p_i$ (profit is the length of the interval)
2. Arbitrary profit

In the first case a reasonable approach is largest processing time (LPT) where the intervals are sorted so that $p_1 \geq p_2 \geq ... \geq p_n$ and then schedule greedily. This does not result in the optimal schedule:



Theorem 1 *LPT is a* $3 - approx$ *algorithm for WISP with proportional profit.*

We show that for proportional profit, LPT is a $3 - approx$ algorithm using a direct charging scheme. This corresponds to finding a scheme where we charge a profit w_i for each $I_i \in OPT$ to some interval $I_j \in LPT$ so that at most $3 \cdot w_j$ gets charged to I_j:
Charge profit of I_i to I_j where $j = argmin_k \{f_k | I_i \bigcap I_k \neq \emptyset, I_k \in LPT\}$



Since $w_i \leq w_j$, else LPT would have taken i before j, the proof holds.

Corrollary 1 *LPT is a* $3\Delta - approx$ *for* $\Delta = \frac{max(w_i/p_i)}{min(w_i/p_i)}$

This raises two obvious questions:

1. Does a greedy $c - approx$ algorithm for proportional profit on a single machine with $c < 3$ exist?
2. Does a greedy $c - approx$ algorithm for WISP with $c < \Delta$ exist?

4 A Precise Model for "Greedy-Like" Algorithms

Before we can answer the above questions we need a precise definition of a greedy algorithm. We try and move from the generally accepted inuition to a useful conception from which we can argue.

We define a "greedy like" algorithm as being a fixed or adaptive priority algorithm as follows [1]:

Fixed Priority

Sort input items $\{I_1, ...I_n\}$ in some manner
for $j = 1..n$ **do**
  make some (greedy) irrevocable decision about I_j
end for

Kruskal's minimum spanning tree algorithm, SPT, LPT and EFT are all examples of fixed priority algorithms.

(Adaptive) Priority

for $j = 1..n$ **do**
  $\mathcal{D} = \emptyset$
  Sort input items based on D and remaining items in I
  make some irrevocable decision about I_{next}
  $\mathcal{I} = \mathcal{I} - \{I_{next}\}$
  $\mathcal{D} = \mathcal{D} \bigcup \{I_{next}\}$
end for

The "natural greedy" set cover algorithm discussed in the first lecture is an example of an adaptive priority algorithm.

To make this more precise:

1. For a given problem, we must say what is the input representation.
2. For a given problem, we must say what are the possible irrevocable decisions.
3. We want to give a more generic condition on what orderings are allowed.

With regard to the ordering, we certainly don't want to allow an arbitrary ordering (otherwise we could find the ordering corresponding to the optimal solution). We would also like to avoid using complexity, e.g. limiting to sorting algorithms which take $O(n \cdot log(n))$. The solution we adopt is that the ordering must be closed under subsets. That is, the relative order of a subset of inputs will remain constant regardless of any changes to input elements not in the subset. This can be accomplished by choosing a priority function from the description of an individual input element to a real number which can then be used to determine the ordering. Ties can be broken arbitrarily, possibly by considering an index in the element description. For fixed priority algorithms the ordering is fixed before any elements are processed. For an adaptive priority algorithm the ordering algorithm can be respecified at each iteration.

Note that no complexity is involved in the above definitions.

5 Analysis of Greedy Algorithms

Given the above formalization of a greedy algorithm we are now prepared to show lower bounds on the $c - approximation$. This comes down to constructing a nemesis input set for all algorithms within the class we are considering. We consider an adversarial proof technique which pits an algorithm against an adversary in a game of the following structure:

1. Adversary presents a large possible set $\mathcal{I}$ of input items.
2. Algorithm indicates how the items would be ordered.
3. Adversary removes some items and presents algorithm with input set $\mathcal{I}' \subseteq \mathcal{I}$

Theorem 2 $c < 3$*, there does not exist a priority algorithm for proportional profit on* $m \geq 1$ *machines with approximation ratio c.*

Theorem 3 $c < \Delta$*, there does not exist a priority algorithm for arbitrary profit with* $m \geq 1$ *machines with approximation ratio less than* Δ*.*

While the above definition is rigorous it does not necessarily completely encompass our intuitive notion of a greedy algorithm. Criticisms of our definition of a greedy algorithm include:

- Sorting based on very small subsets of items not permitted.
- Does not allow the ordering to use conjugate properties such as the total number of items in the input set (the lower bound proof can often be extended to allow such "preprocessing").

5.1 Analysis of $GREEDY_\alpha$

The following algorithm, $GREEDY_\alpha$, is an example of an algorithm which has been called greedy but that does not conform to our definition. This algorithm approximates the solution to the the weighted job interval scheduling problem (WJISP):
$\mathcal{I} = \{I_1, I_2, ..., I_n\}$
$I_j = (s_j, f_j, w_j, c_j)$
$c_j \in \{1, 2, ..., t\}$ is the job to which I_j is scheduled
A feasible schedule can contain only a single interval per job.

$GREEDY_\alpha$ (Erlebach 2000 [2])

1: Sort intervals such that $f_1 \leq f_2 \leq ... \leq f_n$
2: $S = \emptyset$ {set of currently accepted intervals}
3: **for** $i = 1..n$ **do**
4: $\quad C_i =$ minimum-weight subset of S such that $(S \backslash C_i) \bigcup \{i\}$ is feasible
5: $\quad$**if** $w(C_i) \leq \alpha w(i)$ **then**
6: $\quad\quad S = (S \backslash C_i) \bigcup \{i\}$
7: $\quad$**end if**
8: **end for**

$GREEDY_\alpha$ has all appearances of a greedy algorithm but violates our definition by allowing some decisions to be reevaluated. Intervals added to the potential solution may be removed in future iterations if the condition in line 5 finds that the current interval is more promising. Hence, this is neither a fixed nor adaptive priority algorithm.

Theorem 4 *For WJISP* $GREEDY_\alpha$ *is a* $\frac{2}{\alpha \cdot (1-\alpha)}$ *approximation algorithm which is an* $8 - approx$ *when* $\alpha = 1/2$*.*

Theorem 5 *For WISP, $GREEDY_\alpha$ is a $\frac{1}{\alpha\cdot(1-\alpha)}$ approximation algorithm, a $4-approx$ when $\alpha = 1/2$.*

Sketch of approximation bound We begin by defining the following notation:
At the end of the i^{th} iteration let A_i be the current set of intervals in $GREEDY_\alpha$ solution.
Let T_i be all intervals that were in some A_j, $j \leq i$.
$A = A_n = Greedy_\alpha[\mathcal{I}]$

$w(A) \geq (1-\alpha)w(T)$ since interval i is included only if the schedule improves by $(1-\alpha)w_i$. This inequality allows us to consider an indirect charging scheme since a bound on $w(T)$ will imply a bound on $w(A)$.

For every $I_i \in OPT$, charge its w_i profit to intervals in $GREEDY_\alpha$ so that:

- at most $\frac{2w_j}{\alpha}$ gets charged to I_j for WJISP
- at most $\frac{w_j}{\alpha}$ gets charged to I_j for WISP

Then $w(OPT) \leq 2/\alpha \cdot w(T) \leq 2/(\alpha(1-\alpha)) \cdot w(A) = 2/(\alpha(1-\alpha)) \cdot w(GREEDY_\alpha)$

The details of the charging scheme for WJISP_1 (for $m = 1$) proceed as follows: If I_i is in T simply charge w_i to i. Otherwise there was a time you should have taken it but there was a set C_i of conflicting intervals consisting of intersecting intervals Q_i and possibly an interval i' from the same job. $GREEDY_\alpha$ gives us the following inequality:

$$w(C_i) = w(Q_i) + w_{i'} > \alpha \cdot w_i$$

We charge $min\{w_i, 1/\alpha \cdot w_{i'}\}$ to i'. We distribute the remainder evenly among the intervals $\in Q_i$.
So interval $j \in Q_i$ is charged $(w_i - min\{w_i, 1/\alpha \cdot w_{i'}\}) \cdot (w_j/w(Q_i))$ Given $w(Q_i)+w_{i'} > \alpha \cdot w_i$ $j \in Q_i$ is charged at most $1/\alpha \cdot w_j$. Since an interval will only be charged at most once for each of these two cases, the total charge to each interval j in T is at most $2/\alpha \cdot w_j$.

This reasoning can easily be extended for $m > 1$ and WISP.

References

[1] A. Borodin, M. N. Nielsen, and C. Rackoff. (incremental) priority algorithms. In *Algorithmica 37(4)*, pages 295–326, 2003.

[2] T. Erlebach and F. C. Spieksma. Simple algorithms for a weighted interval selection problem. In *Algorithms and Computation: 11th International Conference*, page 228, 2000.