CSC 2421 Approximation Algorithms
Lecture 5, Spring 2005

Lecturer: Allan Borodin
Scribe: Paul McCabe
Department of Computer Science
University of Toronto

1 Adaptive priority stack algorithms

Recall the adaptive priority stack model for packing problems: this model is like the priority model, except that instead of irrevocably accepting or rejecting the next item, it chooses (irrevocably) whether or not to push the next item onto a stack. The final solution is obtained by repeatedly popping the stack: the algorithm must take the next item on the stack if doing so would not produce an infeasible solution.

Input: $S = \{I_1, I_2, \ldots, I_n\}$
  while $S \neq \varnothing$ **do**
    Ordering: choose an ordering on S
    let I_{next} be the first input in this ordering
    Decision: choose whether to push I_{next} onto the stack
    $S \leftarrow S \setminus \{I_{\text{next}}\}$
  end while

  while the stack is not empty **do**
    let I be the top element on the stack; remove I from the stack
    if $S \cup \{I\}$ is feasible **then**
      $S \leftarrow S \cup \{I\}$
    end if
  end while
  return S

Algorithm 1: (Adaptive) priority stack algorithm for packing problems

This model is a generalization of the priority model: to simulate a priority algorithm in the stack model, push an item onto the stack whenever the priority algorithm would accept it. Since the solution computed by the priority algorithm must be feasible, the contents of the entire stack form a feasible solution, and so in the last stage every element of the stack is taken.

1.1 Job bandwidth allocation

The job bandwidth allocation problem (JBAP) is a scheduling problem where each interval has a start time s_i, a finish time f_i, a value v_i, a bandwidth $0 \leq b_i \leq 1$, and a job class μ_i. The bandwidth allocation problem (BAP) is the special case where $\mu_i = i$ for all i (i.e. each job consists of a single interval). In both cases a feasible solution is a set $S \subseteq \{1, \ldots, n\}$ such that for every time t:

$$\sum_{i \in S : t \in [s_i, f_i)} b_i \leq 1$$

In the case of JBAP, there is the additional requirement that $\mu_i \neq \mu_j$ for each $i \neq j \in S$. Many scheduling-type problems can be viewed as special cases of JBAP:

$$\begin{aligned} \text{WJISP:} \quad & b_i = 1 \\ \text{WISP}_1: \quad & b_i = 1,\ \mu_i = i \\ \text{WISP}_k: \quad & b_i = 1/k,\ \mu_i = i \\ \text{KNAPSACK:} \quad & b_i = \text{size of } i\text{-th item},\ s_i = 0,\ f_i = 1,\ \mu_i = i \end{aligned}$$

1.2 Upper bounds for bandwidth-allocation

Last class, we saw a general "local ratio" algorithm for JBAP. The algorithm had a parameter α, and we proved the following theorem bounding the approximation ratio achieved by the algorithm in terms of α and the largest and smallest bandwidths.

Theorem 1.1. *For an instance of JBAP, let v_{opt} be the smallest upper bound on the optimal profit with respect to v_1, and let $v_{\max}$ be the largest lower bound on the approximation ratio achieved by any $\tilde{I}$-maximal schedule. Then the local ratio algorithm achieves an approximation ratio of $v_{\max}/v_{\text{opt}}$. Also, the following bounds hold for $v_{\max}$ and v_{opt}:*

$$\begin{aligned} v_{\max} &\geq \min\{1, \alpha \cdot \max\{b_{\min}, 1 - b_{\max}\} \\ v_{\text{opt}} &\leq 1 + \alpha \end{aligned}$$

where $b_{\min}$ and $b_{\max}$ are the smallest and largest bandwidths in the instance.

Notice that the algorithm LOCAL-RATIO fits into the priority stack model (in fact, it is even a fixed-priority stack algorithm): the intervals are ordered by non-decreasing finish time and an interval is pushed onto the stack if it still has positive value when it is considered. In the final stage, each interval popped from the stack is taken if it is feasible to do so.

For general JBAP (or BAP), theorem 1.1 does not directly provide any bound on the approximation ratio, as $b_{\min}$ and $1 - b_{\max}$ may be arbitrarily small. It does, however, provide bounds for several of the special cases mentioned above.

WISP_1: LOCAL-RATIO with $\alpha = 1$ is the earliest-finishing-time greedy algorithm, which solves this problem optimally.

WISP_k: Recall that for this problem we set each bandwidth to $1/k$. Choosing $\alpha = 1$, this means that we have

$$v_1(I) = \begin{cases} v(\tilde{I}), & \text{if } I = \tilde{I} \\ v(\tilde{I})/k, & \text{if } I \text{ overlaps } \tilde{I} \\ 0, & \text{otherwise} \end{cases}$$

Consider the optimal profit with respect to v_1. Since $\tilde{I}$ has the earliest finishing time, all jobs which overlap $\tilde{I}$ overlap each other at the point $f(\tilde{I})$, so the optimal solution schedules $\tilde{I}$ and $k - 1$ jobs which overlap it, for a profit of

$$v(\tilde{I}) + v(\tilde{I}) \cdot (k-1)/k = v(\tilde{I})(1 + (k-1)/k)$$

On the other hand, every $\tilde{I}$-maximal solution either contains $\tilde{I}$, or contains k intervals which overlap it, and in both cases achieves a profit of at least $v(\tilde{I})$ with respect to v_1. So the approximation ratio achieved by LOCAL-RATIO is at least

$$\frac{1}{1+(k-1)/k} = \frac{k}{2k-1}$$

which tends to $1/2$ for large k. Note that WISP_k can actually be solved optimally in polynomial time, using dynamic programming or network flows.

BAP: Partition the intervals into two sets. I_{big} contains all intervals with bandwidth strictly larger than $1/2$, and I_{small} contains the remaining intervals. The intervals in I_{big} are exactly like an instance of WISP_1, as bandwidths larger than $1/2$ are equivalent to bandwidth 1, and so an optimal schedule S_{big} can be found for these intervals. For I_{small}, consider the profit function v_1 at a point in the algorithm.

$$v_1(I) = \begin{cases} v(\tilde{I}), & \text{if } I = \tilde{I} \\ v(\tilde{I}) \cdot b(I) \cdot \alpha, & \text{if } I \text{ intersects } \tilde{I} \\ 0, & \text{otherwise} \end{cases}$$

As an $\tilde{I}$-maximal solution must contain either $\tilde{I}$, or a collection of intervals that intersect $\tilde{I}$ and have total bandwidth larger than $1 - b(\tilde{I})$. In the former case the v_1-profit is at least $v(\tilde{I})$, while in the latter case the total v_1-profit is at least $v(\tilde{I}) \cdot \alpha(1 - b(\tilde{I}))$. On the other hand, the optimal solution with respect to v_1 either takes $\tilde{I}$ and intervals intersecting $\tilde{I}$ having total bandwidth $1 - b(\tilde{I})$, in which case it achieves a v_1-profit of

$$v(\tilde{I}) + v(\tilde{I}) \cdot \alpha(1 - b(\tilde{I})) = v(\tilde{I})(1 + \alpha(1 - b(\tilde{I})))$$

Or it does not take $\tilde{I}$, in which case the best v_1-profit it can obtain is $v(\tilde{I}) \cdot \alpha$. Choosing $\alpha = 1/(1 - b(\tilde{I}))$, we have that every $\tilde{I}$-maximal solution achieves a v_1-profit of at least $v(\tilde{I})$, while the optimal solution achieves a v_1-profit of at most $2 \cdot v(\tilde{I})$, yielding an approximation ratio of $1/2$. So we are able to obtain S_{big} and S_{small} satisfying

$$v(S_{\text{big}}) = v(\text{OPT}_{\text{big}})$$
$$v(S_{\text{small}}) \geq (1/2) \cdot v(\text{OPT}_{\text{small}})$$

Clearly $v(\text{OPT}_{\text{big}}) + v(\text{OPT}_{\text{small}}) \geq v(\text{OPT})$. First suppose that the following holds.

$$v(\text{OPT}_{\text{big}}) \geq (1/3) \cdot v(\text{OPT})$$

In this case S_{big} is a $(1/3)$-approximation for the whole instance. Otherwise we have

$$v(\text{OPT}_{\text{small}}) \geq (2/3) \cdot v(\text{OPT})$$

in which case S_{small} is a $(1/3)$-approximation for the whole instance. So either S_{big} or S_{small} is a $(1/3)$-approximation. The algorithm simply takes the one that has a higher total value.

Two comments are in order. First of all, the $(1/2)$-approximation for the small intervals uses a different value of α at each stage. This does not affect the analysis of the local ratio algorithm. Secondly, the algorithm we just described is not a stack algorithm: it runs two separate stack algorithms in parallel, and takes the better of the two solutions. More on this later.

JBAP: The idea here is similar to the approach for BAP. Partition the intervals into I_{big} and I_{small} as before. For I_{big}, the problem is equivalent to the case where

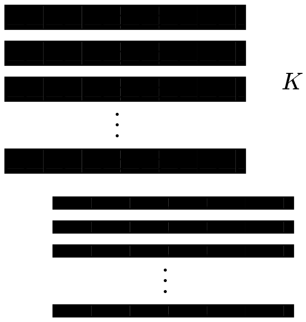

Figure 1: Proof of theorem 1.2

all intervals have bandwidth 1, and theorem 1.1 with $\alpha = 1$ guarantees an approximation ratio of $1/2$ for this case. For I_{small}, theorem 1.1 with $\alpha = 2$ guarantees an approximation ratio of $1/3$. If the optimal value on I_{big} is at least a $2/5$ factor of the overall optimal value, then S_{big} is a $(1/5)$-approximation. Otherwise the optimal value on I_{small} is at least a $3/5$ factor of the optimal value for the whole instance, in which case S_{small} is a $(1/5)$-approximation. We therefore take the better of S_{big} and S_{small} to obtain a $(1/5)$-approximation algorithm.

1.3 Lower bounds for bandwidth allocation

In this section we prove lower bounds for bandwidth allocation problems on the priority stack model.

Theorem 1.2. *For every positive integer k and every $\varepsilon > 0$, when all bandwidths are restricted to being at most $1/k$, and ordering is by finish times, no stack algorithm can achieve an approximation ratio $(1 - 1/k) + \varepsilon$ for BAP.*

Proof. Suppose, to derive a contradiction, that a stack algorithm that orders by finishing times achieves an approximation ratio of $(1-1/k)+\varepsilon$ for some $\varepsilon > 0$. The initial set of intervals is as pictured in figure 1.3, with K "fat" intervals of bandwidth and value $1/K$ and N^2 "slim" intervals of bandwidth $1/N^2$ and value $1/N$. The strategy of the adversary is as follows: if the algorithm rejects a fat interval, then the adversary removes all remaining intervals. In this case the algorithm achieves an approximation ratio of at most $(K-1)/K = 1-1/K$. So assume that the algorithm pushes every fat interval onto the stack. If it pushes a slim interval onto the stack, then the adversary removes all remaining intervals. In this case the algorithm must take the slim interval (since it is on the top of the stack), and can only take $K - 1$ fat intervals, for a total profit of $(K - 1)/K + (1/N)$; the optimal solution takes all the fat intervals and achieves a profit of 1, so the approximation ratio in this case is $(1 - 1/K) + 1/N$. Finally, if the algorithm does not push a slim interval onto the stack, then the adversary does nothing: in this case the algorithm schedules all K fat intervals and achieves a profit of 1, while the optimal solution takes every slim interval and achieves a profit of N. The approximation ratio in this case is $1/N$. To get a contradiction, choose $N > 1/\varepsilon$. $\square$

Theorem 1.3. *For every $\varepsilon > 0$ and every α, the local ratio algorithm (with parameter α) cannot achieve an approximation ratio of $1/2 + \varepsilon$ for BAP.*

To prove this theorem we will use the following property of the local ratio algorithm.

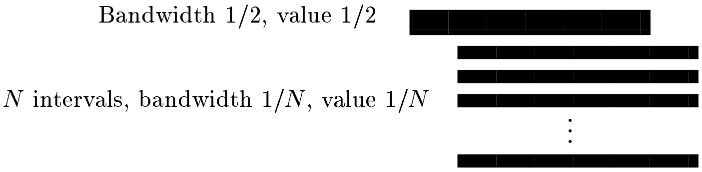

Figure 2: Proof of Theorem 1.3

Lemma 1.4. *If an instance contains many copies of the same interval, then the local-ratio algorithm either pushes none of these intervals, one of them, or all of them onto the stack.*

Proof. Suppose that there exist identical intervals $I_1, \dots, I_k$ with bandwidth b, and that the local ratio algorithm takes at least two of them. We will show that it must take all k. First of all, since the intervals are identical they have the same finish time, so the algorithm considers them consecutively. Also, since at any point in time these intervals have the same value, once the algorithm rejects one of these intervals for the first time (because its value is non-positive) it will also reject the remaining intervals. Suppose the intervals are considered by the algorithm in the order $I_1, \dots, I_k$, and let v be the value of $I_1, \dots, I_k$ at the moment the first interval I_1 is considered. Then the value immediately after I_1 is considered is

$$\begin{aligned} v' &= v - v\alpha b \\ &= (1 - \alpha b)v \end{aligned}$$

Since the algorithm pushes I_2 onto the stack, it follows that v' is positive, and thus $(1 - \alpha b) > 0$. In this case it is not hard to see that after $I_1, \dots, I_j$ have been considered, the value of the remaining intervals is $(1 - \alpha b)^j v > 0$, and thus I_{j+1} will be pushed onto the stack. □

Proof of Theorem 1.3. The initial instance, pictured in figure 1.3, consists of a fat interval with value and bandwidth $1/2$, and N slim intervals with value and bandwidth $1/N$, where N is even. Clearly the algorithm pushes the fat interval onto the stack. By lemma 1.4, it either pushes zero, one, or all of the slim intervals, regardless of how many slim intervals are available. If it does not push any slim intervals onto the stack, then it achieves value $1/2$ while the optimal solution schedules all the slim intervals for a value of 1, so in this case the approximation ratio is $1/2$. If it pushes one slim interval onto the stack, then it achieves a value of $1/2 + 1/N$ and the optimal solution achieves a value of 1, so the approximation ratio is

$$\frac{1}{2} + \frac{1}{N}$$

On the other hand, if it pushes all of the slim intervals onto the stack, then the adversary removes all but $N/2 + 1$ of them. In this case the local-ratio algorithm must accept each of the slim intervals (since they are feasible and are on the top of the stack), and is unable to accept the fat interval since the total bandwidth is $(N/2 + 1)/N = 1/2 + 1/N$. The optimal solution schedules $N/2$ slim intervals and the fat interval to obtain a value of 1, so in this case the approximation ratio is again $1/2 + 1/N$. The theorem follows by choosing N large enough. □

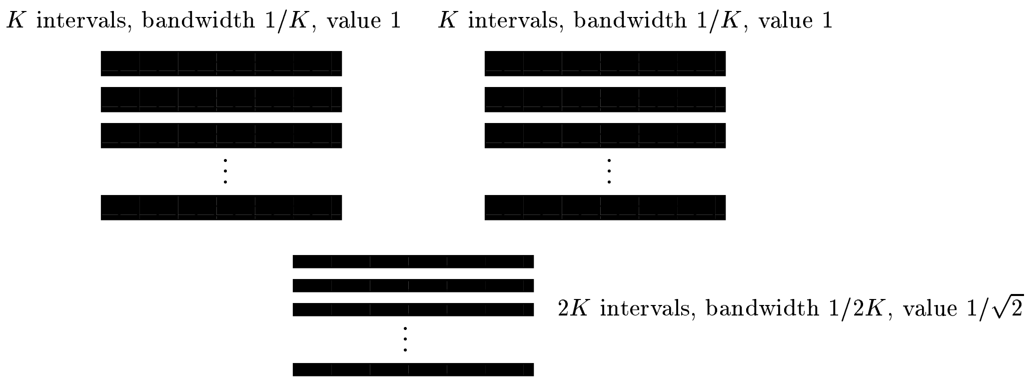

Figure 3: Proof of Theorem 1.5

Theorem 1.5. *For every $\varepsilon > 0$, if all bandwidths are at most $1/k$ then no fixed-priority stack algorithm achieves an approximation ratio for BAP of*

$$1 - \left(\frac{1 - 1/\sqrt{2}}{k} \right) + \varepsilon$$

Proof. The initial instance, pictured in figure 1.3, consists of two sets of K fat intervals, each with bandwidth $1/K$ and value 1; and $2K$ slim intervals, each with bandwidth $1/2K$ and value $1/\sqrt{2}$. Note that a fixed-priority algorithm must consider identical intervals consecutively, so there are three cases.

Case 1*: One of the fat groups is considered first.* If the algorithm rejects a fat interval then the adversary removes all remaining intervals, and the approximation ratio is at most

$$\frac{K-1}{K} = 1 - \frac{1}{K}$$

So assume the algorithm pushes K fat intervals onto the stack. At this point the adversary removes the other group of fat intervals, so the algorithm must consider the slim intervals next. If the algorithm pushes a slim interval onto the stack, the adversary removes the remaining slim intervals. The algorithm must include the slim interval in its schedule (since this interval is on the top of the stack), so it is only able to schedule $K-1$ fat intervals, for a total value of $K - 1 + 1/\sqrt{2}$. The optimal solution takes all K fat intervals for a total value of K. In this case the approximation ratio is

$$\frac{K - 1 + 1/\sqrt{2}}{K} = 1 - \left(\frac{1 - 1/\sqrt{2}}{K} \right)$$

Case 2*: The slim group is considered first.* If the algorithm rejects a slim interval then the adversary removes all the remaining intervals, and the approximation ratio is at most $1 - 1/2K$. So assume that the algorithm pushes every slim interval onto the stack. Next, the algorithm considers one of the fat groups. If it rejects one of these intervals, then the maximum value it can obtain is $2K - 2 + \sqrt{2}$, by scheduling $K-1$ fat intervals from each group and two slim intervals. The optimal

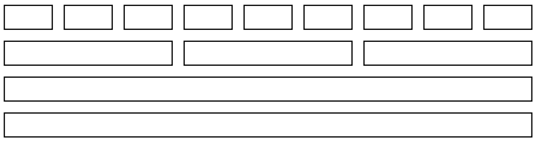

Figure 4: Proof of Theorem 1.6 (All bandwidths are 1/2)

solution achieves a value of $2K$ by taking all the fat intervals, so in this case the approximation ratio is

$$\begin{aligned}\frac{2K - 2 + \sqrt{2}}{2K} &= 1 - \left(\frac{2 - \sqrt{2}}{2K}\right) \\ &= 1 - \left(\frac{1 - 1/\sqrt{2}}{K}\right)\end{aligned}$$

So assume that the algorithm schedules all K fat intervals from the group it is considering. At this point the adversary removes the other fat group. The algorithm must take the K fat intervals it just pushed onto the stack, and is unable to take any slim intervals, so it achieves a value of K. The optimal solution takes all $2K$ slim intervals for a value of $\sqrt{2}K$. The approximation ratio in this case is $1/\sqrt{2}$, which is bounded above by

$$1 - \left(\frac{1 - 1/\sqrt{2}}{K}\right)$$

for all $K \geq 1$. ☐

Theorem 1.6. *For every $\varepsilon > 0$, if all bandwidths are $1/2$, then no fixed-priority stack algorithm achieves an approximation ratio of $13/14 + \varepsilon$ for BAP.*

Note. The case where all bandwidths are $1/2$ corresponds to WISP_2, implying an identical upper bound on the approximation ratio achievable by a fixed-priority stack algorithm for this problem.

Proof. (Sketch) The initial instance is pictured in figure 1.3. We identify the intervals with their values. First of all, for any subinstance, the algorithm must push the first two intervals it examines onto the stack: otherwise, the adversary removes all but the first two intervals and wins. Secondly, the algorithm cannot place a smaller interval between two larger-valued intervals in its ordering: if it does so, the adversary removes all intervals but these three. The algorithm must push the first two intervals (which includes the smaller-valued interval) onto the stack, and its solution, which must contain the smaller interval, achieves a ratio below 13/14.

From the above it follows that the 6 and the 5 are ordered consecutively. In the fixed ordering, either two of the 4's occur before the 6 and the 5, or two of them occur after these intervals. The adversary keeps two 4's that occur on the same side, and deletes the other one. By the same argument, these two 4's must be ordered consecutively, so for each of them, two of the corresponding 3's are on the same side. The adversary keeps two such 3's and deletes the other, for both of the 4's. At this point, there are not many possible orderings remaining, and an inspection of each of them shows that in all cases, the algorithm fails to beat a 13/14 ratio. ☐

2 Towards a new model of computation

Recall that the knapsack problem is a special case of bandwidth scheduling, where all intervals start at time 0 and end at time 1. In this setting the bandwidth corresponds to the size of the item. There is a fixed-priority greedy algorithm that achieves a $1/2$ approximation for the *simple* knapsack problem, where each item has value equal to its bandwidth: the algorithm sorts the items by non-increasing value and chooses items greedily (if an item fits when it is considered, then the algorithm takes it). By contrast, there is no constant-approximation priority algorithm for the *general* knapsack problem (where values may be arbitrary1), as the next theorem shows.

Theorem 2.1. *For every constant $c \leq 1$, there does not exist a priority algorithm that achieves a c-approximation to the general knapsack problem.*

Proof. Consider an initial instance consisting of big, medium, and small items (with many copies of each). The big items have bandwidth 1 and value 1; the medium items have bandwidth $1/n$ and value $1/\sqrt{n}$; and the small items have bandwidth $1/n^2$ and value $1/n$. Consider the first $n - n^{3/4}$ items considered by the algorithm. If the algorithm accepts a small item, then delete all items but the big ones. The algorithm achieves a profit of $1/n$, while the optimal solution achieves a value of 1, so the approximation ratio is $1/n$. If the algorithm accepts a big item, then delete all items but the small ones. The algorithm achieves a profit of 1, while the optimal solution schedules all the small intervals and achieves a profit of n, so the approximation ratio is $1/n$ in this case as well. If the algorithm accepts a medium item, then delete all but the big items. The algorithm achieves a profit of $1/\sqrt{n}$ while the optimal solution takes the big item and achieves a profit of 1, so the ratio is $1/\sqrt{n}$ in this case. Finally, suppose the algorithm does not accept any of the first $n - n^{3/4}$ items it considers. Then the adversary deletes all but $n^{3/4}$ small items. The algorithm achieves a profit of at most $n^{3/4-1} = n^{-1/4}$. If the items rejected by the algorithm included a big item, then the optimal solution achieves a value of 1; if the rejected items included at least $\sqrt{n}$ medium items then the optimal solution achieves a profit of 1, giving an approximation ratio of at most $1/n^{1/4}$. If neither of these situations occurs then the rejected items contain at least $n - n^{3/4} - \sqrt{n}$ small items, and thus there are at least $n - \sqrt{n}$ small items which the optimal solution can use to obtain a profit of at least $1 - 1/\sqrt{n}$. The approximation ratio in this case is $O(n^{-1/4})$. Thus for every n, there is an instance of n items in which the algorithm achieves an approximation ratio of $n^{-1/4}$. □

The "natural" greedy algorithm for general knapsack sorts the items in non-increasing order by "bang per buck", the ratio v_i/b_i of value to bandwidth. As the above theorem shows, this algorithm does not achieve a constant-factor approximation. However, consider the following modification: the first time an item would be rejected, compare the value of this item to the total value contained in the solution obtained so far. If the last item considered has greater value than all the other items together, then select that item; otherwise select the other items.

This modification of the natural greedy algorithm is a 2-approximation for general knapsack. To see this, suppose that the items are ordered so that

$$\frac{v_1}{b_1} \geq \frac{v_2}{b_2} \geq \cdots \geq \frac{v_n}{b_n}$$

Let k be the index of the first item rejected by the natural greedy algorithm. Then the total bandwidth of the first k items exceeds 1, and these are the most efficient items available, so the total value of the first k items is no smaller than the optimal value. Thus either the first $k-1$ items, or the k-th item, has total value of at least half the optimal value. In fact, this argument shows that either the natural greedy algorithm, or the solution consisting only of the largest item, is a $1/2$-approximation.

What kind of algorithm is this? We could view it as a one-pass algorithm with revocable acceptances: if an item would cause overflow but has higher value than the items already in the knapsack, then discard the entire knapsack and take the new item. Or, it is a priority stack algorithm: push items on the stack greedily, keeping track of their bandwidth and value, and the first time an item is considered which would overflow, push that item onto the stack only if it has value greater than the total value of all items already on the stack (and reject all remaining items).

We can also view it as a new type of algorithm: one that maintains several possible solutions, and in the end chooses the solution which achieves the best objective value. For example, consider a new algorithm that generalizes the previous one: take the k largest items, and for every subset of these items, obtain a solution by filling up the remaining bandwidth using the natural greedy method. In the end, take the feasible solution that achieves the highest value. This algorithm requires 2^k solutions to be maintained, and achieves an approximation ratio of $k/(k+1)$.

Definition 1. A uniform approximation scheme for a minimization (maximization) problem is an algorithm which, given an instance I of the problem and $\varepsilon > 0$, returns a $(1+\varepsilon)$-approximate ($(1-\varepsilon)$-approximate) solution. If the algorithm runs in time polynomial in the size of I, then it is called a polynomial-time approximation scheme (PTAS). If it runs in time polynomial in the size of I and in $1/\varepsilon$, then it is called a fully polynomial-time approximation scheme (FPTAS).

The algorithm that considers all subsets of the k largest items is a PTAS for the knapsack problem. Given $\varepsilon > 0$, choose k so that $k/(k+1) \geq (1-\varepsilon)$, i.e. choose $k = 1/\varepsilon - 1$. The running time is polynomial in n, but exponential in $1/\varepsilon$, as $2^k = 2^{1/\varepsilon - 1}$ subsets must be considered.

In this extended model of computation, our lower bounds will depend on the number of solutions which an algorithm is allowed to maintain.

2.1 An FPTAS for the Knapsack problem

Suppose we are given an instance $(b_1, v_1), \ldots, (b_n, v_n), B$ of the general knapsack problem. For $S \subseteq \{1, \ldots, n\}$ let $v(S)$ and $b(S)$ denote the total value and bandwidth of the items indexed by S. There are two common ways to find an optimal solution by dynamic programming. The first way is based on an array $\mathrm{OPT}[i, b]$ defined as

$$\mathrm{OPT}[i, b] = \max\Big\{ v(S) \;\Big|\; S \subseteq \{1, \ldots, i\},\ b(S) \leq b \Big\}$$

The array can be filled in using the recurrence

$$\mathrm{OPT}[i, b] = \max\Big\{ \mathrm{OPT}[i-1, b],\ \mathrm{OPT}[i-1, b-b_i] + v_i \Big\}$$

The values need to be computed up to $\mathrm{OPT}[n, B]$, requiring time $O(nB)$. The second formulation, which is more useful here, is based on an array

$$\mathrm{OPT}'[i, v] = \min\Big\{ b(S) \;\Big|\; S \subseteq \{1, \ldots, i\},\ v(S) \geq v \Big\}$$

which can be filled in using the recurrence

$$\mathrm{OPT'}[i, v] = \min\Big\{ \mathrm{OPT'}[i-1, v],\ \mathrm{OPT'}[i-1, v-v_i] + w_i \Big\}$$

In this case we need to let v range up to $V = \sum v_i$; the optimal value is the largest value of v for which OPT'$[n, v] \leq B$. The idea of the FPTAS is to only use the high-order bits of the values. For large-valued items this is a good approximation to their actual values. Next class we will see the FPTAS in detail, and introduce a formal model of "dynamic programming" that includes this algorithm.

References

[1] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor and B. Schieber. *A unified approach to approximating resource allocation and scheduling.* J. ACM 48(5), pages 1069-1090, 2001.

[2] D. Cashman. *Approximate truthful mechanisms for the knapsack problem, and negative results using a stack model for local ratio Algorithms.* M. Sc. Thesis, University of Toronto, 2005.

[3] S. L. Horn. *One-pass algorithms with revocable acceptances for job interval selection.* M. Sc. Thesis, University of Toronto, 2004.