Approximation Algorithms (p. 68-83)

Petri Savola

Department of Information and Computer Science, TKK

21.2.2008

Outline

- Approximation scheme definitions
- Knapsack
- Strong NP-hardness
- Bin Packing
- Minimum Makespan Scheduling

Basic definitions

- Let Π be an NP-hard optimization (minimization) problem with objective function f_Π. Algorithm A is an approximation scheme for Π if on input (I, ϵ), where I is an instance of Π and $\epsilon > 0$ is an error parameter, it outputs a solution s such that $f_\Pi(I, s) \leq (1 + \epsilon)OPT$.
- A is said to be a polynomial time approximation scheme (PTAS), if for each fixed $\epsilon > 0$, its running time is bounded by a polynomial in the size of instance I.
- If the running time of A is bounded by a polynomial in the size of instance I and $1/\epsilon$ then A is said to be a fully polynomial time approximation scheme (FPTAS).

Knapsack Problem

Given a set $S = \{a_1, ..., a_n\}$ of objects, with specified sizes and profits, $size(a_i) \in \mathbf{Z}^+$ and $profit(a_i) \in \mathbf{Z}^+$, and a capacity $B \in \mathbf{Z}^+$, find a subset of objects whose total size is bounded by B and total profit is maximized.

- An algorithm is said to be pseudo-polynomial if its running time is bounded by a polynomial in $|I_u|$, where $|I_u|$ is the unary size of instance I.
- Knapsack problem allows a pseudo-polynomial time algorithm.
- This fact is used to create a FPTAS for Knapsack.

FPTAS for Knapsack

Let $A(i, j)$ be the maximum profit that can be attained with size less than or equal to j using items up to i.

- $A(0, j) = 0$
- $A(i, 0) = 0$
- $A(i, j) = A(i - 1, j)$ if $size(a_i) > j$
- $A(i, j) = max\{A(i - 1, j), profit(a_i) + A(i - 1, j - size(a_i))\}$ if $size(a_i) \leq j$

Using this algorithm, the solution to the Knapsack problem can be found in time $\mathcal{O}(nB)$ simply by calculating $A(n, B)$.

FPTAS for Knapsack (continued)

- Problem: profits of objects are not bounded by a polynomial in n.
- Solution: ignore some number of least significant bits of profits!
- Result: Profit of at least $(1-\epsilon)OPT$ in time bounded by a polynomial in n and $1/\epsilon$ (Theorem 8.4).

FPTAS for Knapsack (continued)

- Let P be the profit of the most profitable item.
- Given $\epsilon > 0$, let $K = \frac{\epsilon P}{n}$.
- For each object a_i, define $profit'(a_i) = \lfloor \frac{profit(a_i)}{K} \rfloor$.
- Using $profit'$ as the profit function, find the most profitable set S.
- Output S.

Proof of Theorem 8.4

Let O denote the optimal set. It is easy to see that $K \cdot profit'(a_i) = profit(a_i) - C_i$, where $0 \leq C_i < K$. Thus, $K \cdot profit'(O) \geq profit(O) - nK \Rightarrow profit(O) - K \cdot profit'(O) \leq nK$.

The dynamic programming step gives us the optimal solution and thus $profit(S) \geq K \cdot profit'(O) \geq profit(O) - nK = OPT - \epsilon P \geq OPT - \epsilon OPT = (1 - \epsilon)OPT$.

The running time of the algorithm without division by K is $\mathcal{O}(nB)$. A trivial upper bound for this is $\mathcal{O}(n^2 P)$. By ignoring the least significant bits (division by K) we get running time $\mathcal{O}(n^2 \lfloor \frac{P}{K} \rfloor) = \mathcal{O}(n^2 \lfloor \frac{n}{\epsilon} \rfloor)$, which is polynomial in n and $1/\epsilon$, proving the theorem.

Strong NP-hardness

A problem Π is strongly NP-hard if every problem in NP can be polynomially reduced to Π in such a way that numbers in the reduced instance are always written in unary.

- Knapsack is not strongly NP-hard unless P = NP.

Theorem 8.5

Let p be a polynomial and Π be an NP-hard minimization problem such that the objective function f_Π is integer valued and on any instance I, $OPT(I) < p(|I_u|)$. If Π admits an FPTAS, then it also admits a pseudo-polynomial time algorithm.

Proof of Theorem 8.5

Suppose there is an FPTAS for Π whose running time on instance I and error parameter ϵ is $q(|I|, 1/\epsilon)$, where q is a polynomial. Next, set $\epsilon = 1/p(|I_u|)$, and run the FPTAS. By definition it produces a solution with objective function value:

$$(1+\epsilon)OPT(I) < OPT(I) + \epsilon p(|I_u|) = OPT(I) + 1$$

Thus, the solution is optimal and the running time is bounded by a polynomial in $|I_u|$. Therefore we have obtained a pseudo-polynomial time algorithm for Π and the proof is complete.

Bin Packing

Given n items with sizes $a_1, ..., a_n \in (0, 1]$, find a packing in unit-sized bins that minimizes the number of bins used.

First-Fit algorithm is a factor 2 approximation algorithm for bin packing. The idea of this algorithm is to put each item to the first possible bin it fits into and else open a new bin. If the algorithm uses m bins, then at least $m - 1$ bins are more than half full. Therefore,

$$OPT \geq \Sigma_{i=1}^{n} a_i > \frac{m-1}{2} \Rightarrow m - 1 < 2OPT \Rightarrow m \leq 2OPT$$

Theorem 9.2

For any $\epsilon > 0$, there is no approximation algorithm having a guarantee of $3/2 - \epsilon$ for the bin packing problem, assuming P $\neq$ NP.

If we were able to find such an algorithm for bin packing, then we would be able to solve the NP-hard number partitioning problem in the following way:

- Let the bin size be $\frac{1}{2}\Sigma_i a_i$.
- If n items can be packed into 2 bins of that size, then we have a solution for the number partitioning problem.
- Thus, if we had a $3/2 - \epsilon$ approximation algorithm, it would find the optimal packing.

Theorem 9.3

For any ϵ, $0 < \epsilon \leq 1/2$, there is an algorithm A_ϵ that runs in time polynomial in n and finds a packing using at most $(1+2\epsilon)OPT + 1$ bins.

Proof. Let I denote the given instance, and I' denote the instance obtained by discarding items of size $< \epsilon$ from I. By Lemma 9.5 it is possible to find a packing for I' using at most $(1+\epsilon)OPT(I')$ bins. Next, the remaining items ($< \epsilon$ size) are packed in First-Fit manner into the bins. If no additional bins are needed, then a packing in $(1+\epsilon)OPT(I') \leq (1+\epsilon)OPT(I)$ bins has been obtained.

Else, let M be the total number of bins used. All but the last bin must be full to the extent of at least $1-\epsilon$. Therefore, the sum of the item sizes in I is at least $(M-1)(1-\epsilon) \leq OPT$. Hence, $M \leq \frac{OPT}{1-\epsilon} + 1 \leq (1+2\epsilon)OPT + 1$, because $\epsilon \leq 1/2$.

Lemma 9.4

Let $\epsilon > 0$ be fixed, and let K be a fixed nonnegative integer. Consider the restriction of the bin packing problem to instances in which each item is of size at least ϵ and the number of distinct item sizes is K. There is a polynomial time algorithm that optimally solves this restricted problem.

Proof. The number of items in a bin is bounded by $\lfloor 1/\epsilon \rfloor = M$. Therefore, the number of different bin types is bounded by $R = \binom{M+K}{M}$. The number of bins used is at most n and hence the number of possible packings is bounded by $P = \binom{n+R}{R}$, which is polynomial in n (actually $\mathcal{O}(n^R)$). One can enumerate these in polynomial time and pick the optimum.

Lemma 9.5

Let $\epsilon > 0$ be fixed. Consider the restriction of the bin packing problem to instances in which each item is of size at least ϵ. There is a PTAS that solves this restricted problem within a factor of $(1 + \epsilon)$.

Proof. In the book.

Bin Packing PTAS algorithm

- Remove items of size $< \epsilon$.
- Round to obtain constant number of item sizes.
- Find optimal packing.
- Use this packing for the original item sizes.
- Pack items of size $< \epsilon$ using First-Fit.

Note that this algorithm is not any kind of practical solution to the bin packing problem, but in theory it's nice, because it works in polyomial time.

Minimum Makespan Scheduling

Given processing time for n jobs, $p_1, ..., p_n$, 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.

It is easy to find a factor 2 algorithm for the problem. The idea is to schedule the jobs one by one, in any order, and assign each job to the machine with least amount of work so far.

Let $start_j$ be the time when the last job (j) is started. Clearly $start_j \leq \frac{1}{m}\Sigma_i p_i \leq OPT$ and $p_j \leq OPT$, thus $start_j + p_j \leq 2OPT$.

PTAS for minimum makespan

- Minimum makespan problem is strongly NP-hard.
- Thus, it does not admit an FPTAS if P $\neq$ NP.
- The problem can be reduced to bin packing.
  There exists a schedule with makespan t if and only if n objects of sizes $I = \{p_1, ..., p_n\}$ can be packed into m bins of capacity t each.
  Let $bins(I, t)$ represent the minimum number of bins of size t required to pack objects in set I. Then, the minimum makespan is given by $min\{t|bins(I, t) \leq M\}$.

PTAS for minimum makespan (continued)

The idea is the following:

- $LB = max\{\frac{1}{m}\Sigma_i p_i, max_i\{p_i\}\}$.
- Perform a binary search between upper and lower bounds of minimum makespan ($LB \leq OPT \leq 2LB$). This search should be terminated at some point to guarantee polynomial running time.
- There exists a dynamic programming algorithm that solves the restricted bin packing problem in $\mathcal{O}(n^{2k})$ time, where k is the number of object sizes.
- Round the object sizes to gain bounded number of different sizes allowing the use of the bin packing algorithm.

The result of this approach is a valid schedule having makespan at most $(1+3\epsilon)OPT$ in time $\mathcal{O}(n^{2k\lceil log_2 \frac{1}{\epsilon}\rceil})$, where $k = \lceil log_{1+\epsilon}\frac{1}{\epsilon}\rceil$.

Summary

- The idea was to obtain polynomial time algorithms for NP-hard problems such that we can decide the error parameter and gain speed for inaccuracy.
- FPTAS for knapsack
- We showed that a strongly NP-hard problem does not allow an FPTAS
- PTAS for bin packing
- The redcution from minimum makespan problem to bin packing