CSC 363 Summer 2005

Lecture Week 13

Approximation Algorithms

- In an optimization problem, compromise on optmizing the objective function
- Optimization problem:
  - given an input instance *I* (e.g. a graph, a sequence of weights)
  - output an object *S* (e.g. a set of vertices, a subsequence of weights)
  - *S* must satisfy some constraints relating to *I* (e.g. set is a clique, weights selected fit in knapsack)
  - objective function assigns a real number value to every possible output object (e.g. size of the set, weight of items selected)
  - on every instance *I*, output object *S* which optimizes objective function (e.g. minimum size set, maximum weight packed)

Approximation Algorithms

- A polytime *r(n)*-approximation algorithm *ALG* for a minimization problem:
  - *ALG* is a polynomial time algorithm
  - for every instance *I*, let *ALG(I)* denote object output by *ALG* when run on instance *I*
  - for every instance *I*, let *OPT(I)* denote object which achieves optimal solution on *I*
  - $f(OPT(I)) <= f(ALG(I)) <= r(n) * f(OPT(I))$
  - equivalently, $1 <= f(ALG(I)) / f(OPT(I)) <= r(n)$
  - *n* is some parameter, not necessarily size of input instance (e.g. number of vertices in graph)

Approximation Algorithms

- How close can we get to the optimum?
- Equivalently, how close to 1 can *r(n)* be?
- For Vertex Cover, we have seen an approximation with *r(n)* = 2
- Known: there is no approximation for Vertex Cover with *r(n)* <= 1.36 unless P=NP
- For general TSP we proved that we cannot have an approximation with *r(n)* being any constant, unless P=NP.
- There are problems for which we can find arbitrarily good approximations!

Approximation Schemes

- A Polynomial-Time Approximation Scheme (PTAS) for a maximization problem is an algorithm that:
  - takes as input an instance *I*
  - takes as input an extra parameter $\varepsilon > 0$
  - for fixed ε,runs in time polynomial in *n*, the size of *I*
  - outputs a solution *ALG(I)* that satisfies
    $(1-\varepsilon)* f(OPT(I) <= f(ALG(I)) <= f(OPT(I))$
- The behaviour of the algorithm as ε decreases can be "wild". Runtime may be $O(n^{(1/\varepsilon)!})$
- A Fully-Polynomial-Time Approximation Scheme (FPTAS) is an algorithm where the running time is polynomial in both *n* and $1/\varepsilon$. E.g. $O(n^3(1/\varepsilon)^4)$

FPTAS for MAX-GK

- General Knapsack problem:
  - given a sequence of items
  - each has a weight w_i and a profit p_i
  - given a capacity *W*
  - select subsequence of weights such that
    - weight capacity not exceeded: $\Sigma_{i \text{ in } S}\, w_i <= W$
    - profit is maximized: $\Sigma_{i \text{ in } S}\, p_i$
- GKD is NP-complete as SKD <=p GKD
- MAX-GK is the optimization problem

A Pseudo-Polynomial Time Algo for MAX-GK

- Dynamic programming approach
- Define *A[i, j]* = minimum weight subset of *{1…i}* that has profit at least *j* (if no such subset exists, let *A[i, j] = infinity*)
- Let *P* = maximum profit of any single item
- Maximum profit we can hope for is *n*P*
- If we knew *A[n, j]* for every *0 <= j <= n*P*, the maximum profit is the largest *j* such that *A[n, j] ≠ infinity* and *A[n, j] <= W*

A Pseudo-Polynomial Time Algo for MAX-GK

- How to compute *A[i, j]*:
  - *A[i, 0] = empty set*, for *0 <= i <= n*
  - *A[0, j] = infinity*, for *0 < j <= n*P*
  - for *i > 0* and *j > 0*, we have *A[i, j] =*
    - *infinity*, if $A[i-1, j] = A[i-1, j-p_i] = infinity$
    - *A[i-1, j]*, if $w(A[i-1, j]) <= w(A[i-1, j-p_i]) + w_i$
    - $A[i-1, j-p_i] + \{ i \}$, otherwise
- This algorithm is exact (finds the optimum solution)
- Runtime is $O(n^2P)$, which is "pseudo-polynomial", because only *log(P)* is polynomial in input size, not *P* itself.
- Pseudo-polynomial = polynomial in values of numbers in input, not in size of those numbers.

Scaling and Rounding

- Many times, a pseudo-polynomial algorithm leads to a PTAS using scaling and rounding
- If all profits p_i have a common factor *f*, and we divide each profit by this *f*, the optimal solutions remain the same
- If *f* is not a common factor, and we still divide the profits by *f* and round the values to integers, optimal solutions need not be the same, but we hope that they are "close"

FPTAS for MAX-GK

- Given $I = ((w_1, p_1), \ldots, (w_m, p_m), W)$ instance to MAX-GK
- Let *K* be a factor to be determined later
- Define a new instance *I'* by
  $w'_i = w_i$, $p'_i = \lfloor p_i / K \rfloor$, $W' = W$
- Run pseudo-polynomial algorithm on *I'*. This produces a set *S*. Output *S* as an answer to the original instance *I*.
- Note *S* is feasible, because weights are the same
- Runtime: $O(n^2 * P/K)$, where *P* is maximum profit in *I*.

FPTAS for MAX-GK

- Let *O* be optimal solution to instance *I*
- Key idea: since algorithm is exact, *S* is the optimal solution to *I'*, so *p'(S) >= p'(O)*.
- Then:
  $p(S) = \Sigma_{i \text{ in } S}\, p_i = K * \Sigma_{i \text{ in } S}\, (p_i / K) >=$
  $>= K * \Sigma_{i \text{ in } S} \lfloor p_i / K \rfloor = K * \Sigma_{i \text{ in } S}\, p'_i = p'(S)$
- And:
  $p'(O) = K * \Sigma_{i \text{ in } O}\, p'_i = K * \Sigma_{i \text{ in } O} \lfloor p_i / K \rfloor >=$
  $>= K * \Sigma_{i \text{ in } O}\, (p_i / K - 1) = p(O) - K * |O|$
- Therefore, *p(S) >= p(O) – K * n*

FPTAS for MAX-GK

- Assume we throw away items that have a weight greater than *W* before we compute *P*, the maximum profit of a single item
- Then, *p(O) >= P*
- For given slack variable **ε**, let *K = ε*P / n*
- We get
  $p(S) >= p(O) - K * n = p(O) - \varepsilon * P >=$
  $>= p(O) - \varepsilon * p(O) = (1-\varepsilon) * p(O)$
- Runtime $O(n^2 * P/K) = O(n^2 * n/\varepsilon) = O(n^3 * 1/\varepsilon)$

Review Computability

- definitions and properties: TM variants, recognizable, decidable, computable function, mapping reduction
- diagonalization method
- common languages: A_{TM}, E_{TM}, …
- big picture: there are things we cannot solve algorithmically

Review Complexity

- definitions and properties: TM running time, P, nondeterministic TMs, NP, coNP, verifiers, *polytime reductions*, (co)NP-hard, (co)NP-complete, approximation algorithms
- sketch of proof that SAT is NP-complete
- a bunch of reductions
- decision/search/optimization problems
- big picture: NP-hard = likely not polynomial, examples of reductions