CMPSCI 611 Advanced Algorithms **Fall 2000**

Lecture 22: November 30

Lecturer: Micah Adler *Scribe: Zihui Ge, Alicia (Pippin) Wolfe*

22.1 Review of TSP

Christofides [C76]: Proved that the Metric version of TSP (in which the distance function obeys the triangle law) has a 3/2-approximation.

Arora [A96]: Attempted to prove that the 3/2-approximation was the best possible one, and found a *better* approximation for a more restricted version of the problem.

Euclidean version of TSP:
The cities are restricted to a plane and the distances between them are distances on the plane. This problem is **NP**-complete, though the proof is not given here.

For all $\epsilon > 0$, a $1+\epsilon$ approximation exists and can be found in $n^{O(1+1/\epsilon)}$ time.

If ϵ is considered to be constant, this is a polynomial time algorithm in n. However, if ϵ is very small, the exponent in the polynomial may be very large. For example, $\epsilon = 0.01$ yields an exponent of $n^{O(101)}$.

22.2 Polynomial Time Approximation Scheme

Definition 22.1 *A problem has a* polynomial time approximation scheme *(PTAS) if and only if* $\forall \epsilon > 0$ *it has a polynomial time* $(1+\epsilon)$*-approximation.*

Definition 22.2 *A PTAS is called a* fully *polynomial time approximation scheme (FPTAS) if the runtime of the* $(1+\epsilon)$*-approximation is bounded by a polynomial in* $\frac{1}{\epsilon}$.

22.2.1 KNAPSACK

KNAPSACK has a FPTAS algorithm.

The KNAPSACK problem:
Input: A set of items numbered as $1, 2, \ldots, n$;
a weight w_i for each item i;
a value v_i for each item i;
and a knapsack capacity, C.
Output: A subset, B, of the items with the maximum total value s.t. $\sum_{i \in B} w_i \leq C$.

22.2.2 Previous Knapsack Algorithm

In the previous lecture we introduced a dynamic programming scheme to resolve the KNAPSACK problem.

Define $knap(i, w) =$ to be the maximum value obtained using items $1, 2, \ldots, i$ and capacity w.

$Knap(i+1, w) = \max(Knap(i, w), Knap(i, w - w_i + 1) + v_i)$

Computing $knap(i, w)$ row by row in a table of size $n \cdot C$ we can get the solution: $knap(n, C)$.

The runtime of this algorithm is $O(n \cdot C)$, and it will always give exact solution. (Notice that C is exponential to its input size – $\log(C)$.)

22.2.3 New Knapsack Algorithm

Now, we give an alternative Pseudo-polynomial time algorithm for finding the exact solution, then convert it to an approximation algorithm.

This algorithm will attempt to minimize the weight for a specific value, rather than attempting to maximize the value for a specific weight.

Each entry in our dynamic programming table will be the minimum weight for a given value:

Define $Vknap(i, v)$ to be the minimum weight required to achieve a value of at least v using items $1, 2, \ldots, i$. Then we can come up with a formula to compute $Vknap(i, v)$:

$$Vknap(i+1, v) = \min\{ Vknap(i, v), Vknap(i, v - v_{i+1}) + w_{i+1}\}$$

and, for the first row in the table:

$$Vknap(1, v) = \begin{cases} w_1 & \text{if } v \leq v_1 \\ \infty & \text{if } v > v_1 \end{cases}$$

To find the optimum solution, we will examine the last row, $Vknap(n, v)$, and find the largest value achievable, i.e., we will find the largest column, v_{max}, such that $Vknap(n, v_{max}) \leq C$. (Notice that $Vknap(n, v)$ is non-decreasing as v increases.)

To analyze the runtime of this algorithm, we define $V = \max_i(v_i)$. Thus the maximum solution achievable is nV. The size of the table in Figure 22.1 is n^2V. We can see the runtime of this algorithm is $O(n^2V)$.

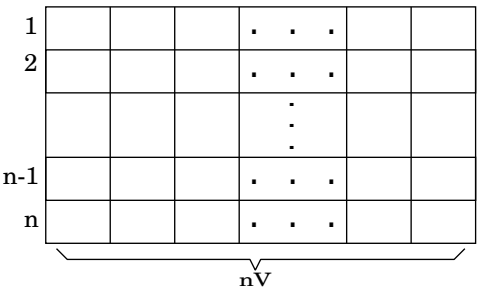

Figure 22.1: $Vknap(i, v)$.

22.2.4 Knapsack Approximation

This algorithm can be easily converted into an approximation algorithm. We can see the intuition from the following example. If we take every v_i and round it down to the closest multiple of 1000, then any

sum achievable from this modified input will be multiple of 1000. This helps to reduce the total number of columns needed in the table (by a factor of 1000). However, the cost of using such a scheme is that it is not guaranteed to find the optimum solution.

From this, we get an approximation for KNAPSACK: Change the k lowest order bits of every v_i to 0 (this is the easiest way to "round down" the inputs). The tradeoff here is that when k is small, our approximation is good (ϵ is small); while when k is large, the runtime is reduced.

Determining k: for FPTAS we need a $1+\epsilon$ approximation. This will give an upper bound on k, we would like the largest k within this bound in order to minimize the running time of the algorithm.

Let $v_i' = v_i$ with the k lowest order bits zeroed.
Let C_{alg} be the answer returned by the modified problem.
Let C_{opt} be the optimal answer to the original problem.
Let B' be the optimal subset of items in the modified problem.
Let B be the optimal subset of items in the original problem.

Since this is a maximization problem, we need to find k such that $\frac{C_{opt}}{C_{alg}} \leq 1+\epsilon$, for a given ϵ.

$$C_{alg} = \sum_{i \in B'} v_i'$$

Since the set B' represents the maximum solution to the modified problem, the solution formed by the set B must be less than or equivalent to it (in the modified problem). Therefore:

$$C_{alg} = \sum_{i \in B'} v_i' \geq \sum_{i \in B} v_i'$$

Since each v_i' has the k smallest bits set to 0, $v_i' > v_i - 2^k$ for all v_i, and

$$\sum_{i \in B} v_i' > \sum_{i \in B} v_i - 2^k \geq C_{opt} - n \cdot 2^k$$

Therefore:

$$\begin{aligned}
C_{alg} &\geq C_{opt} - n \cdot 2^k \\
\frac{C_{opt}}{C_{alg}} &\leq \frac{C_{opt}}{C_{opt} - n \cdot 2^k} \\
&= \frac{C_{opt} - n \cdot 2^k + n \cdot 2^k}{C_{opt} - n \cdot 2^k} \\
&= 1 + \frac{n2^k}{C_{opt} - n2^k} \\
&\leq 1 + \frac{n2^k}{V - n2^k}
\end{aligned}$$

Where V is the maximum v_i, assuming no items exist that cannot fit into the knapsack.

Assuming that $V \geq 2n2^k$, we have

$$1 + \frac{n2^k}{V - n2^k} \leq 1 + \frac{2n2^k}{V}$$

Since we are looking for a $(1+\epsilon)$-approximation, we hope that $\frac{2n2^k}{V}$ will be no larger than ϵ. Thus,

$$\begin{aligned}
\frac{2n2^k}{V} &\leq \epsilon \\
k &\leq \lg(\frac{\epsilon V}{2n})
\end{aligned}$$

To satisfy our previous assumption, it is sufficient that:

$$\begin{aligned} V &\geq 2n2^k \\ &= 2n\frac{\epsilon V}{2n} \\ \epsilon &\leq 1 \end{aligned}$$

which is not a bad assumption since we are interested in using this for some small $\epsilon(\leq 1)$.

22.2.5 Run Time Analysis

Assume that the algorithm removed bits, rather than simply setting them to 0. Each value would then be shifted k bits, and:

the new maximum value is $\frac{V}{2^k}$,

the matrix size is $n \cdot \frac{nV}{2^k}$,

and the run time is $O(\frac{n^2V}{2^k}) = O(\frac{n^2V}{\frac{\epsilon V}{2n}}) = O(\frac{n^3}{\epsilon})$.

which is linear in $1/\epsilon$.

22.2.6 Summary of KNAPSACK

FPTAS for Knapsack problem:

Given a problem instance and an approximation ratio $(1 + \epsilon)$:

1. let $k = \lfloor \log(\frac{\epsilon V}{2n}) \rfloor$;
2. truncate the last k bits from values of all items;
3. solve the smaller problem and return the resulting subset.

22.3 Problems without good approximation algorithms

22.3.1 Review of approximation algorithms

Thus far we have seen, in order of increasing quality of approximation:

- Vertex-Cover/Max-Cut has a 2-approximation,
- Metric TSP has a 3/2-approximation,
- TSP-Euclidean has a PTAS solution, and
- Knapsack has a FPTAS solution.

Is there a FPTAS for every problem?

It is believed that there is no good approximation solution for many problems, however, this is difficult to prove.

If we can prove that there is no FPTAS for any **NP**-complete problem, we can show that $\mathbf{P} \neq \mathbf{NP}$, since any polynomial time algorithm provides an exact solution in polynomial time, and would therefore also provide an FPTAS solution to the problem.

However, we also know that certain problems can't be approximated well unless $\mathbf{P} = \mathbf{NP}$.

22.3.2 Attempting to approximate the general TSP problem

Show: for the general TSP problem, there is no k-approximation for any constant k unless $\mathbf{P} = \mathbf{NP}$.

The proof will be similar to the proof that Metric TSP is **NP**-complete.

Reduce from the Hamiltonian-cycle problem, and using a k-approximation of TSP, find an exact solution for Hamiltonian-cycle.

HamCycle Input: A graph $G = (V, E)$.
TSP Input: cities: V

$$\text{distance function: } d_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ (k+1) \cdot |V| & \text{otherwise.} \end{cases}$$

If G has a Hamiltonian-cycle, the optimal tour in the TSP problem has length $|V|$.

Therefore, with a k-approximation scheme, we can find a solution of length $\leq k \cdot |V|$ if there is a Hamiltonian-cycle.

If there is not a Hamiltonian-cycle, the optimal tour is of length $\geq (k+1) \cdot |V|$ (since it must include at least one edge $(i,j) \notin E$). Therefore, the tour found by the approximation algorithm must also be length $\geq (k+1) \cdot |V|$.

We can then distinguish between

- Case 1: the approximate answer is $\leq k \cdot |V|$, in which case there is a Hamiltonian Cycle, and
- Case 2: the approximate answer is $\geq (k+1) \cdot |V|$, in which case there is no Hamiltonian Cycle.

So, if there is a k-approximation algorithm for TSP then there is a polynomial-time exact solution for the Hamiltonian Cycle problem, which is known to be **NP**-complete, and thus $\mathbf{P} = \mathbf{NP}$.

22.3.3 Types of Approximation Algorithms

In order of decreasing strength:

1. FPTAS - Knapsack
2. PTAS - Euclidean TSP
3. Constant - vertex cover

4. $\log n$
5. n^ϵ, for constant ϵ

22.3.4 Approximating Clique

Example of an n^ϵ, for constant ϵ, approximation:
All solutions to Clique lie between 1 and $|V|$. Therefore, returning a single edge yields a $\frac{|V|}{2}$-approximation.

There is no known good approximation algorithm for this problem. We do know that:

- We cannot approximate this problem within $|V|^{\frac{1}{2}-\epsilon}$ for any $\epsilon > 0$ unless $\mathbf{P} = \mathbf{NP}$.
- We cannot approximate this problem within $|V|^{1-\epsilon}$ for any $\epsilon > 0$ unless every problem in the class **NP** can be solved by a Las Vegas randomized polynomial-time algorithm.

A Las Vegas algorithm never returns the wrong answer and, in this case, has a expected runtime that is polynomial.

References

A96 S. Arora, Polynomial time approximation schemes for Euclidean TSP and other geometric problems, *37th Annual Symposium on Foundations of Computer Science, IEEE Comput. Soc. press.*

C76 N. Christofides, Worst-case analysis of a new heuristic for the traveling salesman problem, *Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh.*