CS880: Approximations Algorithms

Scribe: Tom Watson **Lecturer:** Shuchi Chawla
Topic: Bin Packing and Euclidean TSP **Date:** 2/6/2007

In the previous lecture, we saw how dynamic programming could be employed to obtain an FPTAS for the Knapsack problem. The key idea was to morph the given instance into another instance with additional structure — namely that the item profits weren't too large — that allowed us to solve it exactly, and such that an optimal solution for the morphed instance could be used to construct a near-optimal solution for the original instance. In this lecture, we will further explore this idea by applying it to the Bin Packing and Euclidean TSP problems. For the Bin Packing problem, our morphed instanced will have a solution space that is small enough to search exhaustively. For the Euclidean TSP problem, we will place geometric contraints on the morphed instance that allow us to solve it exactly using dynamic programming.

5.1 Bin Packing

5.1.1 The Problem

The Bin Packing problem is, in a sense, complementary to the Minimum Makespan Scheduling problem, which we studied in a previous lecture. In the latter problem, the goal is to schedule jobs of various lengths on a fixed number of machines while minimizing the makespan, or equivalently to pack items of various sizes into a fixed number of bins while minimizing the largest bin size. We now consider the problem where we swap the roles of constraint and objective: all bins have a fixed size, and we wish to minimize the number of bins needed.

Definition 5.1.1 (Bin Packing) *Given items with sizes $s_1, \ldots, s_n \in (0, 1]$, pack them into the fewest number of bins possible, where each bin is of size 1.*

Note that the assumption that the bins are of size 1 is without loss of generality, since scaling the bin size and all item sizes by the same amount results in an equivalent instance.

It is easy to see that Bin Packing is NP-hard by a reduction from the following problem.

Definition 5.1.2 (2-Partition) *Given items with sizes $s_1, \ldots, s_n$, can they be partitioned into two sets of equal size?*

Clearly, an instance of 2-Parition is a yes-instance if and only if the items can be packed into two bins of size $\frac{1}{2}\sum_{i=1}^{n} s_i$. Thus a polynomial-time algorithm for Bin Packing would yield a polynomial-time algorithm for 2-Partition. In fact, even a $(3/2 - \epsilon)$-approximation algorithm for Bin Packing would yield a polynomial-time algorithm for 2-Partition: on no-instances it would clearly use at least three bins, but on yes-instances it would use at most $(3/2 - \epsilon)2 < 3$ bins.

Theorem 5.1.3 *For all $\epsilon > 0$, Bin Packing is NP-hard to approximate within a factor of $3/2 - \epsilon$.*

Corollary 5.1.4 *There is no PTAS for Bin Packing unless $P = NP$.*

The above result exploited the fact that OPT could be small. Can we do better if OPT is large? The answer is yes. We will obtain an *asymptotic* PTAS, where we only require a $(1+\epsilon)$-approximate solution if OPT is sufficiently large.

Definition 5.1.5 *An* asymptotic PTAS *is an algorithm that, given $\epsilon > 0$, produces a $(1+\epsilon)$-approximate solution provided $OPT > C(\epsilon)$ for some function C, and runs in time polynomial in n for every fixed ϵ.*

In particular, we will obtain the following result.

Theorem 5.1.6 *There is an algorithm for Bin Packing that, given $\epsilon > 0$, produces a solution using at most $(1+\epsilon)OPT + 1$ bins and runs in time polynomial in n for every fixed ϵ.*

Corollary 5.1.7 *There is an asymptotic PTAS for Bin Packing.*

Proof: Given $\epsilon > 0$, running the algorithm from Theorem 5.1.6 with parameter $\epsilon/2$ yields a solution using at most $(1 + \epsilon/2 + 1/OPT)OPT$ bins, which is at most $(1+\epsilon)OPT$ provided $OPT > 2/\epsilon$. ■

The following algorithm is due to W. Fernandez de la Vega and G. Lueker [4].

5.1.2 The Algorithm

We seek to prove Theorem 5.1.6. Given an instance I of Bin Packing, we would like to morph it into a related instance that can be solved optimally, and for which an optimal solution allows us to construct an near-optimal solution for the original instance I. Our strategy will be to reduce the solution space so it is small enough to be searched exhaustively. One idea is to throw out small items, since intuitively the large items seem to be the bottleneck in finding a good solution. Another idea is to make sure there aren't too many different item sizes. The following result confirms that these ideas accomplish our goal.

Theorem 5.1.8 *There is a polynomial-time algorithm that solves Bin Packing on instances where there are at most K different sizes of items and at most L items can fit in a single bin, provided K and L are constants.*

Proof: Call two solutions equivalent if they are the same up to the ordering of the bins, the ordering of the items within each bin, and the distinguishing of items of the same size. We will show that there are polynomially many nonequivalent solutions, and thus an optimal solution can be found by exhaustive search.

The number of configurations for a single bin is at most K^L, even if we distinguished between different orderings of the items, since a configuration can be specified by the size of each of the at most L items in the bin. The important thing is that it is a constant.

If we are not careful about only counting nonequivalent solutions, we might reason that since a solution uses at most n bins, each of which can be in one of at most K^L configurations, there are at most $(K^L)^n$ solutions. This bound is not good enough, and we can do better by remembering that it only matters how many bins of each configuration there are, not what order they're in. If x_i denotes the number of bins with the ith configuration, then nonequivalent solutions correspond

to nonnegative integral solutions to the equation

$$x_1 + x_2 + \cdots + x_{K^L} \leq n,$$

of which there are at most

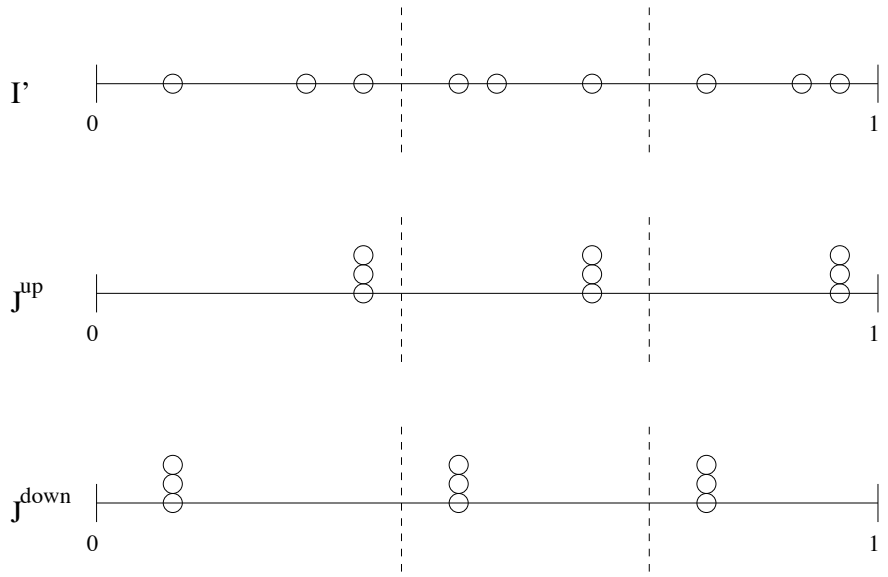
$$\binom{n + K^L}{K^L}$$

by a classic combinatorial argument. This bound is at most a polynomial of degree K^L. ■

In light of the previous theorem, we pause to remark that in contrast to the clever FPTAS for Knapsack we saw in the last lecture, the running time of the algorithm we are developing for Bin Packing is prohibitively expensive. Typically, PTAS's have a bad dependence on $1/\epsilon$. For this reason, PTAS's are not usually very practical and are of primarily theoretical interest.

Given instance I, we seek to morph it into an instance where Theorem 5.1.8 applies. We will first obtain an instance I' by throwing out all items of size less than ϵ. We will worry about packing these items later. Now at most $L = \lceil 1/\epsilon \rceil$ items can fit into any one bin.

There could still be as many as n different item sizes, so we need to morph I' further to get an instance with a constant number K of different item sizes. One way to do this is to consider the items in sorted order, partition them into K groups, and round each item's size up to the largest size in its group, yielding an instance J^{up}. Another way is to round each item's size down to the smallest size in its group, yielding an instance J^{down}.



Neither of these two possibilities seems ideal. We want our new instance to satisfy the following two (informal) properties.

(1) Given a solution to the new instance, it is easy to construct a comparable solution to I'.

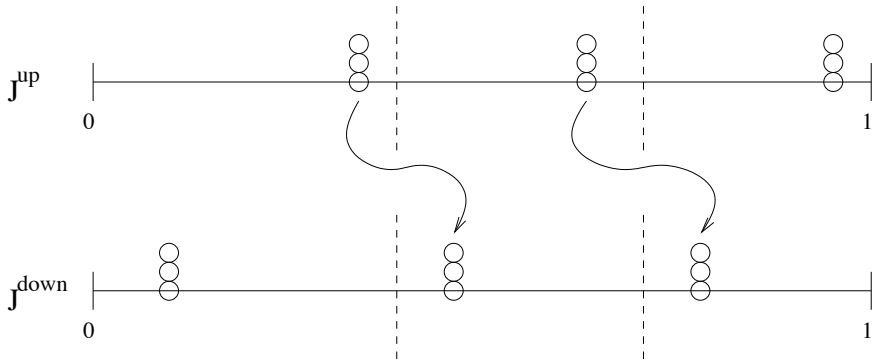
(2) The optimum value of the new instance isn't too much worse than the optimum value of I'.

The instance J^{up} satisfies (1), since when we go back to instance I', the items can only shrink, which means that the same solution is still feasible. However, it doesn't seem to satisfy (2), since if we try to translate the optimal solution for I' into a solution for J^{up}, all the bins could overflow, requiring many new bins to be opened up. The instance J^{down} satisfies (2) since the optimal solution for I' immediately yields a feasibile solution for J^{down} with the same number of bins. However, it doesn't seem to satisfy (1) since a solution to J^{down} can't generally be translated to a solution for I' without lots of bins overflowing.

It turns out that if we select the parameters in the right way, J^{up} *does* satisfy property (2). We can argue this by comparing J^{up} to J^{down}. (However, our final algorithm will apply the algorithm from Theorem 5.1.8 only to J^{up}, not to J^{down}.)

We consider the items in sorted order and partition them into $K = 1/\epsilon^2$ groups of size $Q = n\epsilon^2$ each , breaking ties arbitrarily. (The last group might have fewer items.) We tacitly ignore the pedantic details associated with rounding these quantities to integers, as these details distract from the essense of the algorithm. We obtain J^{up} by rounding each item's size up to the size of the largest item in its group, and similarly obtain J^{down} by rounding each item's size down to the size of the smallest item in its group. For each of these instances, there are at most K different item sizes.

Our algorithm will actually construct J^{up} and apply the algorithm from Theorem 5.1.8 to it. The resulting solution is also a feasible solution for I', as noted above. We would like to show that the number of bins this solution uses is not too much more than $OPT(I')$. As usual, we will need a lower bound on $OPT(I')$ to compare with. Observe that $OPT(J^{\text{down}}) \leq OPT(I')$ since each feasible solution of I' is also a feasible solution of J^{down}. We will use this lower bound. How much worse than $OPT(J^{\text{down}})$ can $OPT(J^{\text{up}})$ be? The critical observation is that a solution to J^{up} can be constructed from a solution to J^{down} by taking each group of items, except the last, and moving them to the locations occupied by the items in the next group, and assigning each item of the last group to its own new bin. Since all groups (except possibly the last) have the same size, this correspondence can be made.



Since the size of every item in a group in J^{up} is at most the size of every item in the next group in J^{down}, it follows that every item of J^{up} is at most the size of the item of J^{down} whose place it's taking. (Note that the locations of the items of the first group of J^{down} aren't filled by any items of J^{up}.) This shows that the resulting solution of J^{up} is feasible. Moreover, it has at most Q additional bins, one for each item in the last group. We conclude that

$$\begin{aligned} OPT(J^{\text{up}}) &\leq OPT(J^{\text{down}}) + Q \\ &\leq OPT(I') + Q \\ &= OPT(I') + n\epsilon^2 \\ &\leq OPT(I') + OPT(I')\epsilon \\ &= (1+\epsilon)OPT(I'). \end{aligned}$$

In going from the first line to the second, we used our lower bound on $OPT(I')$. In going from the third line to the fourth, we use a *second* lower bound on $OPT(I')$, namely that no solution can do better than to pack every bin completely, which would use at least $n\epsilon$ bins (since every item is of size at least ϵ). This reveals why we chose Q the way we did: to make sure that the number of extra bins we used in our comparison of $OPT(J^{\text{up}})$ to $OPT(J^{\text{down}})$ was at most $\epsilon OPT(I')$.

With this result in hand, we can prove the main result of this section.

Proof of Theorem 5.1.6: We are given instance I and $\epsilon > 0$. First, we construct I' by throwing out all items of size less than ϵ, and then we construct J^{up} and solve it optimally using the algorithm of Theorem 5.1.8. The resulting solution, we have argued, is a feasible solution to I' using at most $(1+\epsilon)OPT(I')$ bins. The running time so far is

$$O(n^{K^L}) = O(n^{O(1/\epsilon)^{O(1/\epsilon)}}) = poly(n).$$

But we still have to pack the items of size less than ϵ. For this, we can make use of the empty space in the bins used by our current packing. A natural thing to do is use a greedy strategy: pack each item of size less than ϵ into the first bin it fits in, only opening a new bin when necessary. We next argue that this does the job.

If the greedy phase does not open any new bins, then the number of bins is at most $(1+\epsilon)OPT(I') \leq (1+\epsilon)OPT(I)$ as shown above. Here we used the trivial lower bound $OPT(I') \leq OPT(I)$. If the greedy phase does open a new bin, then at the end, all but the last bin must be more than $1-\epsilon$ full (since otherwise the item that caused the last bin to be opened would have fit into one of them). Denoting by ALG the number of bins used by our algorithm's solution, we conclude that

$$(1-\epsilon)(ALG - 1) \leq \sum_{i=1}^{n} s_i \leq OPT(I).$$

Here we have used a second lower bound on $OPT(I)$. It follows that

$$ALG \leq \frac{1}{1-\epsilon} OPT(I) + 1.$$

We have $\frac{1}{1-\epsilon} = 1 + O(\epsilon)$ provided ϵ is at most some fixed positive constant, which is no loss of generality. Since we may run this algorithm with a smaller ϵ parameter than the one we are given, this suffices to prove the theorem. ■

It's worth noting what prevents this approach from giving a PTAS (instead of an asymptotic PTAS). In the final greedy phase, we can't say anything about how full the last bin to be opened is. This prevents us from applying the $\sum_{i=1}^{n} s_i \leq OPT(I)$ bound to this last bin. Thus we get an extra $+1$ term floating around, which as a fraction of OPT, only goes down as OPT goes up, not as ϵ goes down.

There are a number of heuristics for Bin Packing that give good worst case performance. See [3] for a survey.

Finally, we remark that the practical importance of the Bin Packing problem spawned research into algorithms for generalizations of this problem. For example, one can consider packing higher-dimensional items into higher-dimensional bins. This generalization presents tricky issues not present in the one-dimensional case that we considered. In the 2-dimensional case under certain restrictions on the packing, one can get an asymptotic PTAS [2]. For higher (but still constant) dimensions, constant factor approximations are known. However, we will not explore these results in this course.

5.2 Euclidean TSP

5.2.1 The Problem

In this section we consider the following practically important special case of the traveling salesperson problem (TSP).

Definition 5.2.1 (Euclidean TSP) *Given n points in the d-dimensional Euclidean metric space (for some fixed d), find a minimum length tour that visits all of them.*

For simplicity, we will restrict our attention to the 2-dimensional case $d = 2$. The algorithm we will present generalizes easily to higher dimensions.

The Metric TSP problem, for which we obtained a 3/2-approximation algorithm in a previous lecture, is known to be APX-hard. Euclidean TSP in the plane is NP-hard, but it is conceivable that the special structure of the Euclidean case allows us to overcome the obstacle that prevents us from obtaining a PTAS for Metric TSP. We will show that this is, in fact, the case.

Theorem 5.2.2 *There is a PTAS for the Euclidean TSP problem.*

This result was proved independently by Arora [1] and Mitchell [5]. We will present Arora's algorithm and analysis.

We pause to emphasize the distinction between the problem at hand and the restriction of TSP to planar metrics. A planar metric is one that arises as the shortest path metric of a planar graph. The edge weights on this planar graph can be selected in any way, and need not have anything to do with Euclidean distance. There is also a PTAS for the TSP on planar metrics, but it is quite different from the algorithm we will present.

5.2.2 The Algorithm

Following the theme of the Bin Packing result and the Knapsack result from the previous lecture, our overall strategy will be to morph the given instance into a related instance that has additional structure that allows us to solve it optimally, and which allows us to construct a "good" solution for the original instance from the optimal solution for the morphed instance. In short, we will first modify the instance by moving each point a little bit so that it is in a convenient location, and then we will further modify the instance by imposing some geometric constraints on the tours. We will be able to solve the new instance exactly by dynamic programming, and then construct a tour for the original instance without the cost growing by too much.

Before getting to the details, we first make the simplifying assumption that the smallest bounding square of the given points has side length exactly n^2. This is without loss of generality since we can scale the given instance without affecting its solution set in any significant way. We denote the resulting instance by I. We also observe that the smallest bounding square must have two points on opposite sides (either one on the left side and one on the right side, or one on the bottom and one on the top). Since every tour must traverse the distance from one of these points to the other and back, we get the following lower bound on the optimum, which will be useful later.

Lemma 5.2.3 *$OPT(I) \geq 2n^2$.*

Now we describe the first modification we will make to our instance. We round the coordinates of each input point to integers values, yielding an instance I'. We can argue that this modification doesn't prevent us from getting a good approximation.

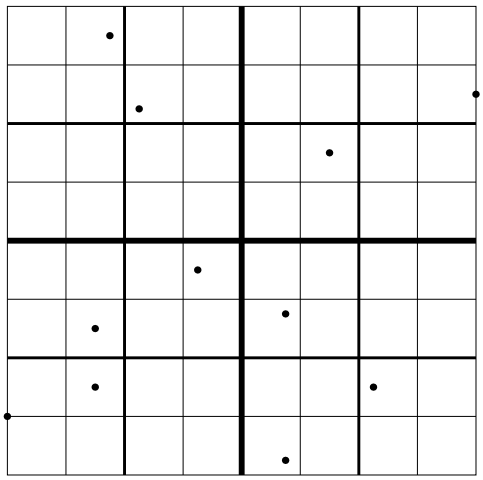
Lemma 5.2.4 *If I' can be approximated within factor $1+\epsilon$, then I can be approximated within factor $1+\epsilon+4/n$.*

Proof: Given a tour of I' of cost $ALG' \leq (1+\epsilon)OPT(I')$, it suffices to show that the corresponding tour of I is of cost $ALG \leq (1+\epsilon+4/n)OPT(I)$. Note that each point in I is at most $\sqrt{2}$ distance from its location in I'. Now given a tour in one of these two instances, the tour in the other instance that follows the same path but "sidesteps" at each input point to visit it's new location and come back has additional total cost at most $2\sqrt{2}n$ and is at least as long as the tour that visits the input points along straight line paths. It follows that corresponding tours in the two instances can differ in cost by at most $2\sqrt{2}n$. Hence, $OPT(I') \leq OPT(I) + 2\sqrt{2}n$ and $ALG \leq ALG' + 2\sqrt{2}n$, and thus

$$\begin{aligned}
ALG &\leq (1+\epsilon)OPT(I') + 2\sqrt{2}n \\
&\leq (1+\epsilon)(OPT(I) + 2\sqrt{2}n) + 2\sqrt{2}n \\
&= (1+\epsilon)OPT(I) + (2+\epsilon)2\sqrt{2}n \\
&\leq (1+\epsilon)OPT(I) + (2+\epsilon)\sqrt{2}\frac{OPT(I)}{n} \\
&\leq (1+\epsilon+4/n)OPT(I).
\end{aligned}$$

We have used Lemma 5.2.3 in going from the third line to the fourth. In going from the fourth line to the fifth, we have assumed that $\epsilon \leq 2\sqrt{2}-2$, which is no loss of generality. Thus given a $(1+\epsilon)$-approximate solution to I', the corresponding solution to I is $(1+\epsilon+4/n)$-approximate. ■

We would like to attempt to solve our morphed instance exactly by dynamic programming. A natural line of attack is to break up the bounding square into four equal parts, and try to recombine solutions to these four subproblems into a solution for the original problem. Each of these subproblems would be broken up into four more subproblems in a similar way, and so on, leading to a 4-ary tree of subproblems.



This motivates the modification we made earlier of rounding all coordinates to integer values. We would like our base case to be when there's just a single point, and this modification ensures that our tree of subproblems won't have to be too deep in order to separate two points that are close to each other.

There seems to be a problem with this naive approach: it's not clear how recombine optimal tours for the four subproblems into an optimal tour for the subproblem at hand. For example, we could have the pathological case where the optimal tour zigzags across one of the dividing lines. Our subproblems need to take into account how they interact with each other across the dividing lines.

On each dividing line we will introduce some number of equidistant *portals* and only consider *portal-proper* tours: ones that only cross dividing lines at portals. Then for each node in the 4-ary tree of subproblems, we will actually have many subproblems, corresponding to different ways of entering and exiting the square at its portals. This increases the number of subproblems, but by setting the parameters properly, we will be able to keep the number under control. This also allows us to form the optimal solution to the subproblem at hand by trying all possible ways of specifying how its four subproblems interact with each other at the portals, and stitching the solutions together. We will require that the number of portals be small enough that we can do this quickly, but large enough that the optimum tour length doesn't deteriorate by too much when we impose this geometric restriction.

More details on this construction will be provided in the next lecture.

References

[1] S. Arora. Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and other Geometric Problems. In *FOCS*, 1996, pp. 2-12.

[2] N. Bansal, A. Lodi, and M. Sviridenko. A Tale of Two Dimensional Bin Packing. In *FOCS*, 2005, pp. 657-666.

[3] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation Algorithms for Bin Packing: A Survey. In *Approximation Algorithms for NP-Hard Problems*, Dorit S. Hochbaum (editor), PWS Publishing Company, 1997, pp. 46-93.

[4] W. Fernandez de la Vega and G. Lueker. Bin Packing Can Be Solved within $1 + \epsilon$ in Linear Time. In *Combinatorica*, 1(4), 1981, pp. 349-355.

[5] J. S. B. Mitchell. Guillotine Subdivisions Approximate Polygonal Subdivisions: A Simple Polynomial-Time Approximation Scheme for Geometric TSP, k-MST, and Related Problems. In *SIAM Journal on Computing*, 28, 1999, pp. 1298-1309.