IOE 691: Approximation & Online Algorithms

Lecture Notes: Dynamic Programming

Instructor: Viswanath Nagarajan Scribe: Gian-Gabriel Garcia, Miao Yu

A third algorithmic technique in approximation algorithms is dynamic programming. Dynamic programming (DP) involves solving problems incrementally, starting with instances of size one and working up to instances of generic size n. It is similar to the method of induction in proofs. A key step in DP is to identify a recursive (or inductive) structure that helps reduce one instance of size n into several instances of size at most $n-1$.

1 Basic Dynamic Programming

We first illustrate some exact DP algorithms.

1.1 Stable Set

The input to the stable (or independent) set problem is given by an undirected graph $G = (V, E)$ and weights $w : V \to \mathbb{R}_+$.

Definition 1.1 *A set of vertices S is called independent if no pair of vertices in S share an edge.*

The goal is to find an independent set S which maximizes the sum of the weights in S.

Here we show a simple dynamic program that solves the problem exactly on *trees*. Results for many graph problems on trees often extend to larger classes of graphs (eg. planar); so trees are a natural special class of graphs to consider. We consider the tree G to be rooted at some vertex r, where all other nodes are descendants of r (i.e., appear below r in the tree).

The recursive problems. Let G_v be the subtree below any vertex v; so $G = G_r$. Let $T[v, 0]$ be the maximum weight of the independent set in G_v such that v is not included in the independent set. Similarly, let $T[v, 1]$ be the maximum weight of the independent set in G_v such that v is included. The following DP gives an exact optimal solution:

Algorithm 1 Dynamic programming algorithm for Stable Set

Re-number vertices by $1, ..., n$ such that the index is increasing from any leaf to the root. So the root of G is vertex n. Let C_i denote all the children of vertex i.
Initialize $T[u, 0] = 0$ and $T[u, 1] = w(u)$ for all $u \in V$ which are leaves.

1. For all non-leaf vertices $i = 1, ..., n$:
   Set $T[i, 0] = \sum_{k \in C_i} \max\{T[k, 0], T[k, 1]\}$. *Remember the maximizing choices.*
   Set $T[i, 1] = w(i) + \sum_{k \in C_i} T[k, 0]$.
2. The optimal value is given by $\max\{T[n, 0], T[n, 1]\}$ at the root.

The optimal solution can also be found by maintaining the maximizing choices (which represent the best decisions) in calculating T, and "backtracking" from the root entry $\max\{T[n, 0], T[n, 1]\}$.

1.2 Longest Path in DAG

As another example of dynamic programming we consider the longest path problem on directed acyclic graphs (DAGs). The input is given by a DAG $G = (V, E)$ and two vertices $s, t \in V$. The objective is to find the $s - t$ path with the maximum number of nodes in it.

Since G is a DAG, a topological ordering for the vertices can be created as:

Algorithm 2 DAG Topological Ordering

1. Set $v_1 = s$. Remove s and all edges attached to it from G.
   Set $i \leftarrow 2$.
2. Do until any vertex remains:
   Find a vertex v with *zero* in-degree.
   Set $v_i \leftarrow v$.
   Set $i \leftarrow i + 1$.
   Remove v and all any edge attached to it from G.

The recursive problems. Assume that all nodes are re-numbered via Algorithm 2. Let

$$T[i] := \text{ length of the longest path from } s \text{ to } v_i, \quad \forall i \in [n].$$

The following DP algorithm can be used to solve the longest path problem exactly.

Algorithm 3 Dynamic programming for longest path in DAG

1. Initialize $T[1] = 1$.
2. For $i = 2, ..., n$:
$$T[i] \leftarrow 1 + \max_{j:(v_j, v_i) \in E} T[j].$$

2 Knapsack

The knapsack problem is concerned with picking a maximum-value subset from n items, each with *value* v_i and *size* s_i, into a knapsack with capacity B (for sizes). We assume that all values/sizes are integer. In homework 1, we derived a polynomial time approximation scheme (PTAS). Using dynamic programming, we can derive a fully polynomial time approximation scheme (FPTAS), which is more efficient. In the FPTAS, for any $\epsilon > 0$, we can get a $(1 - \epsilon)$ approximation in time polynomial in N and $\frac{1}{\epsilon}$, where $N = n \times \log \max_{i \in [n]} \{v_i, s_i\}$ is the input size. On the other hand, a PTAS has runtime polynomial in $N^{1/\epsilon}$; while this is a polynomial for any constant ϵ, it is much slower than the FPTAS runtime. First, we will show that the knapsack problem can be solved exactly using dynamic programming in "psuedo polynomial" time $poly(n, \max_i v_i)$.

The recursive problems. Let $V := \sum_{i=1}^{n} v_i \leq n \cdot v_{max}$ denote the maximum objective value. Consider the related problem where given a target number u, we want to find the minimum size subset of total value at least u. We can guess the optimal value OPT (at most V many choices)

and set $u = OPT$ to solve the original knapsack problem. To achieve target value u, we want to solve the following optimization problem:

$$\min_{I \subseteq [n]} \sum_{i \in I} s_i \qquad (KS^*)$$
$$\text{s.t.} \sum_{i \in I} v_i \geq u$$

It turns out that (KS^*) can be solved exactly using dynamic programming. Let

$$T[i,u] := \text{ minimum size of subset } I \subseteq \{1,...,i\} \text{ such that } \sum_{k \in I} v_k \geq u, \qquad \forall i \in [n], u \in [V].$$

Algorithm 4 Dynamic programming algorithm for (KS^*)

1. For $w = 0, ..., V$:
$$T[1,w] \leftarrow \begin{cases} 0 & w = 0 \\ s_1 & 1 \leq w \leq v_1 \\ \infty & \text{otherwise.} \end{cases}$$
2. For $i = 2, ..., n$:
   For $u = 0, ..., V$:
$$T[i,u] \leftarrow \min\{T[i-1,u], T[i-1,u-v_i] + s_i\}.$$

Finally, to find the optimal solution to the knapsack problem, it suffices to find

$$\max\{w : T[n,w] \leq B\}.$$

Using Algorithm 4, we have shown the following result:

Theorem 2.1 *Assuming that all values are integer, there exists an exact algorithm for the knapsack problem in time that is polynomial in n and v_{max}.*

Note that the knapsack problem is NP-hard! So we cannot expect an exact algorithm in polynomial of n and $\log v_{max}$. We will now use this DP on a modified instance to obtain an FPTAS.

2.1 FPTAS for Knapsack

We assume that there is no item with size greater than B, because we cannot choose such items in an optimal solution. Now, recall that we stated an exact algorithm for Knapsack problem with running time $poly(n, v_{max})$. We refer to this algorithm as $ExactKnapsack$.

Here we provide an FPTAS for the knapsack problem. The main idea is to "round" values into a polynomial range (rather than v_{max} which is psuedo-polynomial). Doing so, we will effectively ignore tiny values at a small ϵ-factor loss in the objective.

Let $M \leftarrow \max_i v_i$ and

$$v_i' \leftarrow \left\lfloor \frac{v_i}{\epsilon M/n} \right\rfloor, \qquad \forall i \in [n].$$

We run $ExactKnapsack$ on the instance with sizes $\{s_i\}_{i=1}^n$ and values $\{v_i'\}_{i=1}^n$ and capacity B. Let $A \subseteq [n]$ be the solution obtained. We will return A as the FPTAS solution.

Theorem 2.2 *The above algorithm is an FPTAS for the knapsack problem.*

Proof: Note that the new values v' are integer and $\sum_{i=1}^{n} v'_i = \sum_{i=1}^{n} \lfloor \frac{v_i}{\epsilon M/n} \rfloor \leq \sum_{i=1}^{n} \frac{n}{\epsilon} \leq \frac{n^2}{\epsilon}$. So, the running time of *ExactKnapsack* on the rounded instance is $poly(\frac{n^2}{\epsilon}) = poly(n, \frac{1}{\epsilon})$, which means the running time of our FPTAS is also $poly(n, \frac{1}{\epsilon})$.

Now, we will show that if OPT denotes the optimal value of the original instance (sizes s_i and values v_i) then our solution A has value at least $(1-\epsilon)OPT$. Let OPT' denote the optimal value of the rounded instance. Let S be the subset of items in the optimal solution under values $\{v_i\}_{i\in[n]}$. We have:

$$OPT = \sum_{i\in S} v_i = \sum_{i\in S} \frac{v_i}{\frac{\epsilon M}{n}} \frac{\epsilon M}{n} \leq \sum_{i\in S} (v'_i + 1)\frac{\epsilon M}{n} = (\sum_{i\in S} v'_i)\frac{\epsilon M}{n} + \epsilon M \leq \frac{\epsilon M}{n} OPT' + \epsilon OPT$$

Where the last inequality is true because $OPT \geq \max_i v_i = M$ as any single item fits in the knapsack (by our assumption). Now by rearranging inequalities we have $OPT' \geq (1-\epsilon)\frac{n}{\epsilon M} \cdot OPT$. It follows that $\sum_{i\in A} v'_i = OPT' \geq (1-\epsilon)\frac{n}{\epsilon M} \cdot OPT$. So,

$$\sum_{i\in A} v_i \geq \frac{\epsilon M}{n} \sum_{i\in A} v'_i = \frac{\epsilon M}{n} OPT' \geq (1-\epsilon)OPT.$$

This completes the proof. ■

3 Euclidean Traveling Salesman Problem

In the Euclidean Traveling Salesman Problem, there are n points in $\mathbb{R}^d$ space with Euclidean distance between any two points, i.e. $d(x, y) = \|x - y\|_2$. We are tasked to find a tour of minimum length visiting each point. In this lecture, we only focus on the case that $d = 2$.

For the metric Traveling Salesman Problem (TSP), there cannot be any polynomial-time approximation scheme (unless P=NP). For a long time, the best approximation bound for metric TSP was 1.5 given by [Chr76]; very recently, this has been improved to $1.5 - \delta$ for some (tiny) constant $\delta > 0$. However, we can design a better approximation algorithm for TSP on Euclidean metrics. In this lecture, we will provide a dynamic programming (DP) scheme for Euclidean TSP that for any $\epsilon > 0$ finds a $(1 + \epsilon)$-approximate solution in polynomial time. This algorithm was developed by [Aro98].

Theorem 3.1 *There is a Polynomial-Time Approximation Scheme (PTAS) for Euclidean Traveling Salesman Problem.*

The strategy to design the PTAS for Euclidean TSP is as follows. Firstly, we round the instance and exploit the structure of the problem to construct a new simpler instance having close-to-optimal objective. Secondly, we design a DP to find an exact solution for the new instance. To simplify the notation, we use OPT to denote both optimal solution and optimal objective value of the Euclidean TSP. We also use the equivalent view of a TSP tour as an Eulerian graph (i.e. connected and even degree everywhere).

3.1 Assumptions

To design our PTAS, we make four important assumptions. First, we notice that scaling the input instance does not affect approximation ratio since the cost of all tours are scaled by same factor. We apply the following scaling on the original input. Let $L = 4n^2$ and scale the input such that $L \times L$ is the smallest square containing all input points. Without loss of generality, we focus on the input of Euclidean TSP after scaling. We make following assumption.

Assumption 3.1 *All input points are on* $\mathbb{Z}_L \times \mathbb{Z}_L$ *with* $L = 4n^2$ *and* $OPT \geq L$

We make following transformation to ensure Assumption 3.1. For each input point $x = (x_1, y_1)$ in original input, we "round" it down to the point $x' = (\lfloor x_1 \rfloor, \lfloor x_2 \rfloor))$. Figure 1 provides an example of transformation with $L = 4$.

Algorithm Rounding

For each point v in(x_v, y_v) from $1, 2, \ldots, n$

$v' \leftarrow (\lfloor x_v \rfloor, \lfloor y_v \rfloor))$

Create an instance with the collection of v''s

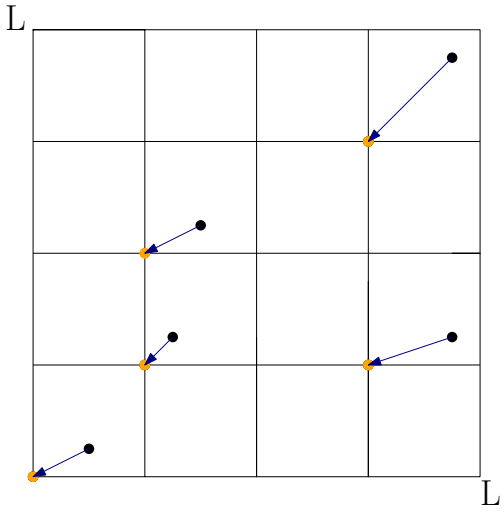

Figure 1: "Rounding" transforms all input points to integer node in $L = 4$ network

Note $OPT \geq L$ is trivial since $L \times L$ is the smallest square containing all input points. Let P_1 denote the instance after "rounding" and OPT_1 be its optimal objective. The following lemma bounds the effect of the transformation.

Lemma 3.1 $OPT_1 \leq (1 + \frac{1}{n})OPT$

Proof: Let v be any original point and v' be its correspondent in the grid network. We find a feasible solution to P_1 from OPT by repeating following process for all points: after visiting point v in OPT, go to v' and then go back to v. The total increase in such solution to visit point v' from v is bounded by $2\sqrt{2}$ and hence overall cost increase is bounded by $2\sqrt{2}n$. Since we have $L \geq OPT$ and $L = 4n^2$, we have relative error no greater than $\frac{2\sqrt{2}n}{OPT} = \frac{2\sqrt{2}n}{L} = \frac{2\sqrt{2}n}{4n^2} < \frac{1}{n}$. Since OPT_1 is the optimal objective, it is no greater than the cost of such solution. Therefore, we conclude $OPT_1 \leq (1 + \frac{1}{n})$. ■

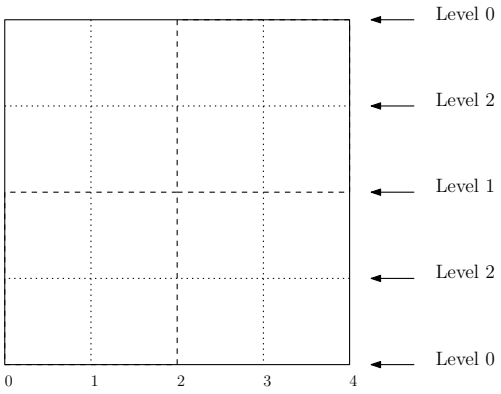

Figure 2: Dissection for the graph with $L = 4$

Next we do a **dissection** of the $L \times L$ square. Starting from $L \times L$ square, we recursively partition each square into 4 equal smaller squares, each of length equal to half the larger square. We continue this process until the length of the square is 1. We define a square to be of level i if the length of its side is $\frac{L}{2^i}$. There are 4^i squares of each level $i \geq 0$. We also define a level for each line (vertical and horizontal). The sides of the $L \times L$ square are of level 0. We define a line to have level $i \geq 1$ if it is used to divide some level $i-1$ square into 4 level i squares. Note that there are 2^{i-1} vertical (resp. horizontal) lines of each level $i \geq 1$. Figure 2 shows an example dissection with $L = 4$.

Consider a square at level i, we define **portals** as certain special points on the sides of the square. On each side of the square, there are m *uniformly spaced* portals: for a total of $4m$ portals around the square. Note that the distance between consecutive portals is $\frac{L}{m2^i}$. We also assume that m is a power of two: so a portal of a level i square is also a portal of any smaller square (i.e. of level at least $i+1$) which is adjacent to its sides. We will eventually set $m \approx \frac{1}{\epsilon}$. Figure 3 shows an example with 16 portals in a square.

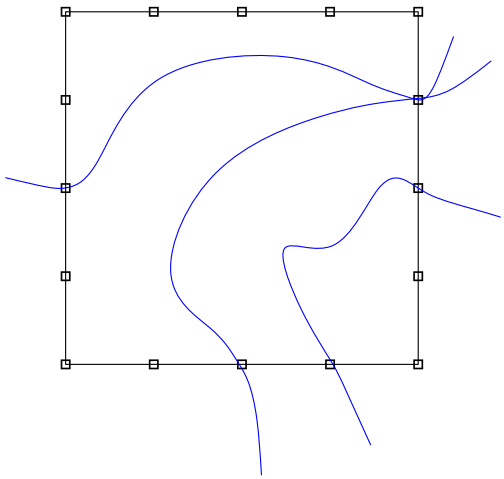

Figure 3: Example of portals. Blue curves denote the portion of tour inside the square.

Assumption 3.2 *The tour enters and exits each square only through portals.*

Assumption 3.3 *The tour enters/exits through each portal no more than $c = O(1)$ times.*

We will view each portal as comprising of c *mini-portals* that are located very close to each other. We then require that each mini-portal is crossed at most once.

Finally, we will require that the tour does not intersect itself.

Assumption 3.4 *The tour is non self-intersecting when all entry/exit at squares occur at mini-portals.*

We will show later that enforcing Assumptions 3.2-3.4 on the optimal tour only increases cost by a small amount.

3.2 DP Algorithm under Assumptions

Here we provide an exact algorithm under Assumptions 3.1-3.3.

Consider a square at level i. By Assumption 3.2 and 3.3, there are $4m$ portals for the square. Since each portal can be crossed at most c times, by considering each portal as c distinct mini-portals the number of entry/exit ways for each square is 3^{4cm} (each of the $4cm$ mini portals has 3 possible states, i.e, in, out or unused). Next, we need to pair the entries and exits noticing that not all pairings will be valid.

Since there is no self-intersection in the tour (Assumption 3.4), we only consider "valid parings" where the enter/exit pairs do not create crossings within the square.

Lemma 3.2 *Given a fixed crossing setting (ways to enter/exits the portals), the number of valid parings is bounded by* $O(2^{4cm})$

Proof: Let $N(2t)$ be the number of parings with $2t$ entry/exit portals. Then after fixing a paring $(1, j)$ for the first entry/exit portal, the number of possible parings is $N(j-2)N(2t-j)$. By summing over all possible choice of j, We have $N(2t) \leq \sum_{j=2}^{2t} N(j-2)N(2t-j)$ where the sum is only over even j. We also have $N(0) = 1$. This can be bounded by the *Catalan number* $C(t)$ because Catalan numbers also satisfy the relationship $C(0) = 1$ and $C(t+1) = \sum_{i=0}^{t} C(i)\,C(t-i)$ for $t \geq 0$. Therefore we conclude the number of valid parings for each entries/exits settings for a square at level i is bounded by $C(2t) = O(2^{2t}) = O(2^{4cm})$. ■

We now build the DP to calculate each OPT for modified network. We define the state of DP as a square and its possible ways to enter/exit such square. Since at each level i, we have 4^i squares, and there are $\log(L) + 1 = O(\log n)$ levels. Then the size of states space is $O(4^L 3^{4cm} 2^{4cm})$ which is polynomial as long as $m = O(\log n)$. The Bellman equation (DP recursion) involves computing the DP entry for a level i square with entry/exit pairing σ using the entries for its 4 level $i-$ squares. This can be done by enumerating over all possible entry/exist pairings $\omega_1, \omega_2, \omega_3, \omega_4$ for the smaller squares and checking of they are compatible with σ (eg. each entry of an ω_k must be either an entry of σ or an exit of another $\omega_{k'}$, there should be no subtours etc). This can be easily done in time polynomial in m.

Therefore, we can find the optimal tour by applying such a DP in $poly(n) \cdot O(6^{4cm})$ time.

Theorem 3.2 *Dynamic Programming given above can solve Euclidean TSP with Assumptions 3.1-3.4 to optimality with* $O(n^4 6^{4cm})$ *time*

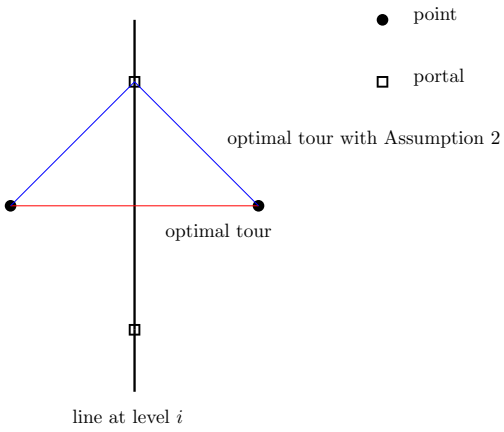

Figure 4: Example of a detour for Assumption 3.2

3.3 Analysis

Now we bound the losses incurred by making Assumptions 3.1-3.4. By Lemma 3.1, we only lose a factor $1 + \frac{1}{n}$ by making Assumption 3.1.

To enforce Assumption 3.2, for each square crossing, we create a detour to its nearest portal. See Figure 4 for an example. Note that for any crossing on a level i *line*, we need to detour to a "level i" portal (i.e. on a square of level i) as this is the strictest requirement (any portal on a level i square is also one on any level $j \geq i+1$ square). Since the distance between two adjacent portals on line at level i is $\frac{L}{m2^i}$, the increase of the overall cost for this detour is at most $\frac{L}{m2^i}$. We define $x(q)$ to be the number of crossings through line q (either horizontal or vertical) in an optimal tour. Let $i(q)$ be the level of line q. After enforcing Assumption 3.2, by summing over all detours, we obtain the following.

Lemma 3.3 *The increase in length of an optimal tour due to Assumption 3.2 is at most* $\sum_q x(q)\frac{L}{m2^{i(q)}}$

Now we look at the effect of the Assumption 3.3. We only give an informal proof here. We argue that the number of crossings is at most $c = 4$. Consider any portal with t crossings (i.e. t entry and t exit points). Consider an infinitesimal circle around the portal and let C denote the $2t$ entry/exit points on this circle. By removing the edges of the optimal tour inside the circle, we obtain a graph with vertices being the original input points, portals and C where (i) all vertices in C have degree one and all other vertices have even degree; and (ii) if C is contracted (representing the portal itself) then the graph will be connected. We now add some edges within the circle to ensure the graph becomes Eulerian with at most c crossings: this would imply the existence of a tour of no larger cost. The edges A to add are (i) a matching on C and (ii) a cycle on C, both of which are in the cyclic order of these points. The cost increase is infinitesimal because all these edges are contained in the infinitesimal circle around the portal. Moreover, the resulting graph is Eulerian. Finally, the resulting tour crosses each portal (on horizontal or vertical line) at most 3 times: note that these crossings will correspond to certain added edges A. See Figure 5.

To enforce Assumption 3.4, we remove self-intersections one at a time. Consider first a self-intersection at a non-portal point p. Consider an infinitesimal circle around this point and let (s_1, t_1) and (s_2, t_2) be the (portions) of the edges through this circle that cause the intersection at p. Suppose the optimal tour is $t_1 \to D_1 \to s_2 \to t_2 \to D_2 \to s_1 \to t_1$ where D_1 (resp. D_2) is a path from t_1 to s_2 (resp. t_2 to s_1). Then the modified tour $t_1 \to D_1 \to s_2 \to s_1 \to D_2^{reverse} \to t_2 \to t_1$

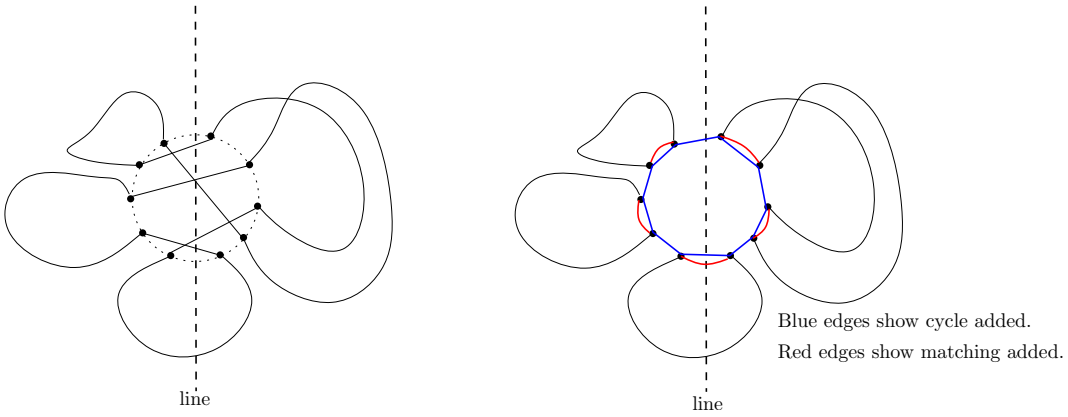

Figure 5: Example of reducing portal crossings ($t = 5$ crossings reduced to 3 crossings).

has no self-intersection and costs less than the original tour; so this needs to be repeated only finite number of times. We refer to this as *uncrossing*. See Figure 6.

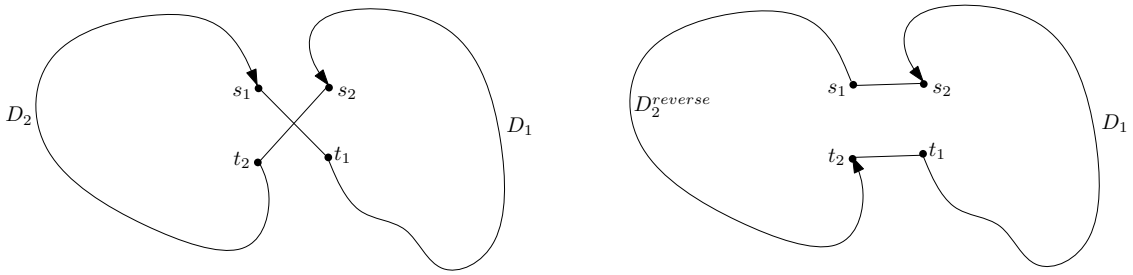

Figure 6: Eliminating self-intersection (non portal).

Now consider self-intersections at a portal. Consider again an infinitesimal circle around the portal and let let $(s_1, t_1), \cdots (s_c, t_c)$ denote the (portions of) all edges of the tour through this circle. By adjusting these edge slightly, we can ensure that there are at most two edges intersecting at any point. Then we perform uncrossing as for non-portal intersections (above) sequentially for each intersection point in the infinitesimal circle. It can be shown that this procedure terminates after a finite number of uncrossings. Again, there is no cost increase by this modification. See Figure 7 for an example.

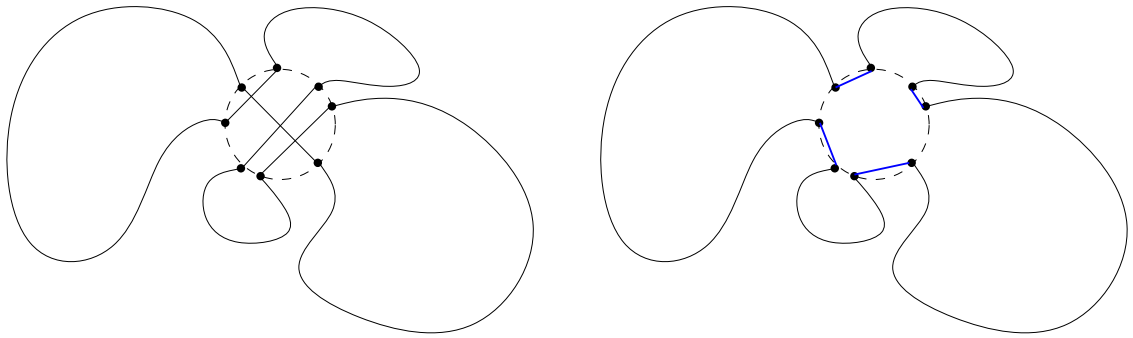

Figure 7: Eliminating self-intersection (portal).

3.4 Randomization to bound cost increase

Now it only remains to bound the cost increase (from Lemma 3.3) in terms of the optimal cost. Recall that the increase is bounded by

$$\frac{L}{m} \sum_{q:line} x(q) \cdot 2^{-i(q)}.$$

Unfortunately, this might even be much larger than OPT as the layout of the points may be such that the optimal tour crosses a line of low level (eg. level 1) several times at non-portal points. A clean idea to handle this issue is via *randomization*.

In **randomized dissection**, we first randomly select integers a, b in range $[-L, 0]$. Then we consider a $2L \times 2L$ grid with "bottom left" at the random integer point (a, b) and "top right" at $(a + 2L, b + 2L)$. Note that irrespective of the (a, b) choice, the integer points $\mathbb{Z}_L \times \mathbb{Z}_L$ (and hence the optimal TSP tour on the given instance) are contained in the resulting $2L \times 2L$ grid. Crucially, we now define the levels of lines and squares in the dissection w.r.t. the randomly shifted square

$$\mathbb{Z}_{[a,a+2L]} \times \mathbb{Z}_{[b,b+2L]}.$$

In particular, for a fixed line q (vertical or horizontal), we have

$$\Pr_{a,b}[\text{ line } q \text{ has level } i] \leq \frac{2^i}{L}, \qquad \forall i = 1, 2 \cdots \log_2(2L).$$

See Figure 8.

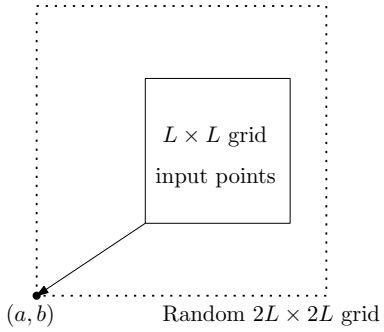

Figure 8: Randomized dissection.

We are now ready to bound the *expected* increase in cost using Lemma 3.3.

Lemma 3.4 *Suppose* $m \geq \frac{\log_2(2L)}{\epsilon}$. *Then the expected cost increase*

$$E_{a,b}\left[\frac{L}{m} \sum_{q:line} x(q) \cdot 2^{-i(q)}\right] \leq \epsilon \sum_{q:line} x(q).$$

Proof: Using the observations (above) on the random dissection, for any fixed line q,

$$E_{a,b}[x(q) \cdot 2^{-i(q)}] \leq x(q) \sum_{i=1}^{\log_2(2L)} 2^{-i} \cdot \Pr_{a,b}[i(q) = i] \leq x(q) \sum_{i=1}^{\log_2(2L)} \frac{1}{L} = \frac{\log_2(2L)}{L} \cdot x(q).$$

Now, adding this over all lines q, we obtain:

$$E_{a,b}\left[\frac{L}{m}\sum_{q:line} x(q)\cdot 2^{-i(q)}\right] \le \frac{L}{m}\cdot\frac{\log_2(2L)}{L}\cdot\sum_{q:line} x(q) \le \epsilon\sum_{q:line} x(q),$$

where the last inequality uses the assumption $m \ge \frac{\log_2(2L)}{\epsilon}$. ■

Finally, we relate the total number of line crossings $\sum_{q:line} x(q)$ to the optimal cost.

Lemma 3.5 $\sum_{q:line} x(q) \le 4\cdot OPT$.

Proof: Note that the optimal tour consists of several edges connecting pairs of points in the $\mathbb{Z}_L \times \mathbb{Z}_L$ grid. Consider any edge from point (x_1, x_2) to (y_1, y_2) in OPT. The distance of this edge is $d(x,y) = \sqrt{(x_1-y_1)^2+(x_2-y_2)^2}$ and the number of line crossings for this edge is at most $|x_1-y_1|+|x_2-y_2|+2$. Since $\alpha+\beta \le \sqrt{2(\alpha^2+\beta^2)}$ for any $\alpha,\beta \ge 0$, we have

$$|x_1-y_1|+|x_2-y_2|+2 \le \sqrt{2((x_1-y_1)^2+(x_2-y_2)^2)}+2 \le 4\sqrt{(x_1-y_1)^2+(x_2-y_2)^2}$$

Summing over all edges in OPT completes the proof. ■

Finally, combining the above two lemmas, the expected increase in cost in order to ensure Assumptions 3.1-3.4 is at most $4\epsilon\cdot OPT$, when $m = \Theta(\log n/\epsilon)$. Thus we obtain an algorithm running in time $n^{O(1/\epsilon)}$ that provides an *expected* $1+\epsilon$ approximation ratio for any $\epsilon > 0$.

To obtain a deterministic algorithm, we can simply try all choices of a, b and run the DP algorithm for each. Therefore we obtain Theorem 3.1.

References

[Aro98] Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.

[Chr76] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group.*, 1976.