CSCE-658 Randomized Algorithms

Lecture #6, February 18, 2016

Lecturer: Professor Jianer Chen

6 The path problem and color-coding

In this section, we first study the PATH problem. The problem is NP-hard because the well-known NP-hard problems HAMILTONIAN-PATH and TRAVELING-SALESMAN are special cases of the PATH problem. Recent research in areas such as bioinformatics has shown interests in solving the PATH problem for short paths. For instance, the research in biological pathways is interested in paths of length no more than 10 in a graph [18]. If we are only interested in short paths, then it becomes possible that the PATH problem is solved effectively. More specifically, we study *parameterized algorithms* for the PATH problem for paths of length bounded by a small integer k, with running time of the form $O(f(k)n^c)$, where c is a constant. The challenge here is to develop parameterized algorithms where the function $f(k)$ is as small as possible. Since the problem is NP-hard, we would not expect that $f(k)$ is a polynomial of k.

We first present a simple randomized algorithm for the parameterized PATH problem, based on random permutation. The technique by itself is interesting and has other applications. Then we introduce the technique of *color-coding*, which is a more recent randomization technique and has proven to be very useful, in particular for developing faster algorithms for NP-hard problems.

6.1 The MAX-PATH problem

A *path* in a graph G is a sequence $\{w_1, w_2, \ldots, w_k\}$ of vertices such that for each i, $[w_i, w_{i+1}]$ is an edge in G. The path is *simple* if no vertex repeats in the sequence. If G is a directed graph, then we require that $[w_i, w_{i+1}]$ be a directed edge from w_i to w_{i+1}, written as $\langle w_i, w_{i+1}\rangle$. A *$k$-path* in G is a simple path of k vertices. If the graph G is weighted, then a *maximum k-path* is a k-path in G whose weight is the maximum over all k-paths in G. In this section, we will be focused on the following problem:

> MAX-PATH. Given an undirected and weighted graph G, and an integer k, construct a maximum k-path, or report no k-paths exist in G.

There is an unweighted version of the problem, where we simply treat all edges as having unit weight, and the problem is asking the existence of k-paths in the graph G. When $k = n$, the unweighted version becomes the HAMILTONIAN-PATH problem. Moreover, the TRAVELING-SALESMAN problem (on undirected graphs) can be easily reduced to the weighted version of the problem where we let $k = n$. Finally, the problem can also be defined on directed graphs. The techniques we describe in this section on undirected graphs can be used to solve the problem on directed graphs.

We first observe that the MAX-PATH problem on directed acyclic graphs is actually easy, which can be solved using the following algorithm based on dynamic programming.

Algorithm 9 DAGPATH
Input: a directed, acyclic, and weighted graph $\hat{G}$, and an integer k;
Output: a maximum k-path in $\hat{G}$;

1. topologically sort the vertices of $\hat{G}$: v_1, v_2, ..., v_n;
2. **for** $i = 1$ **to** $n - 1$ **do**
   $W[i, 1] = 0$; $X[i, 1] = *$;
   for $h = 2$ **to** k **do** $\{W[i, h] = -\infty$; $X[i, h] = *\}$;
3. **for** $i = 1$ **to** $n - 1$ **do**
   for each edge $\langle v_i, v_j\rangle$ **do**
   for $h = 1$ **to** $k - 1$ **do**
   if $W[i, h] \neq -\infty$ and $W[i, h] + \text{wt}[v_i, v_j] > W[j; h + 1]$
   then $W[j, h + 1] = W[i, h] + \text{wt}[v_i, v_j]$; $X[j, h + 1] = i$;
4. The vertex v_t with the largest $W[t, k]$ is the last vertex of a maximum k-path.

We give some explanations on the algorithm DagPath. Each vertex v_i is associated with $2k$ values $W[i,h]$ and $X[i,h]$, where $1 \le h \le k$, where $W[i,h]$ is the weight of the largest h-path $P(h,v_i)$ ending at v_i that has been discovered so far, and $X[i,h]$ is the vertex just before v_i on the path $P(h,v_i)$. Initially at step 2, every vertex v_i only sees the 1-path starting and ending at itself, so it correctly sets $W[i,1] = 0$. Since the 1-path has only one vertex, we let $X[i,1] = *$, which will be irrelevant. We use $W[i,h] = -\infty$ for $h > 1$ to indicate that no h-path ending at v_i has been discovered, yet.

Since the vertices are topologically sorted (so edges only go from vertices of smaller index to vertices of larger index), in the i-loop in step 3 when we reach the vertex v_i, all vertices on any path ending at v_i have been processed, so $W[i,h]$ and $X[i,h]$, for $1 \le h \le k$, record correctly the information for maximum h-path ending at v_i, for all $h \le k$. Then the process in step 3 extends the correct information on v_i to later vertices via the edges from v_i. Therefore, at the end of step 3, the information for the maximum h-path ending at each vertex, for all $h \le k$, is obtained, and the vertex v_t with the largest $W[t,k]$ is the ending vertex of a maximum k-path in the graph G. This maximum k-path $\{w_1, w_2, \ldots, w_k\}$ can be re-constructed, starting from v_t, by following the links given by the array $X[*,*]$ of previous vertices on the path: $w_k = v_t$, $w_{k-1} = X[t,k] = v_i$, $w_{k-2} = X[i,k-1] = v_j$, $w_{k-3} = X[j,k-2]$, and so on.

The running time of the algorithm DagPath is $O(k(n+m))$, where n and m are the number of vertices and the number of edges in the graph $\hat{G}$, respectively.

Now we return back to our original max-Path problem on undirected and weighted graphs. Unfortunately, there seems no easy way to convert an undirected graph G into an "equivalent" directed acyclic graph, even when the undirected graph G itself has no cycles (e.g., G is a tree). Here is where randomization proves useful.

Suppose that the vertex set of the undirected graph G is $V = \{v_1, v_2, \ldots, v_n\}$. Let $\pi = (v'_1, v'_2, \ldots, v'_n)$ be a permutation of $\{v_1, v_2, \ldots, v_n\}$. if we convert each edge $[v'_i, v'_j]$ of G, where $i < j$, into a directed edge $\langle v'_i, v'_j \rangle$ from v'_i to v'_j (and keep the same edge weight), then G becomes a directly acyclic graph $\hat{G}_\pi$. The paths in $\hat{G}_\pi$ have the following properties:

P1. Every (directed) path in $\hat{G}_\pi$ corresponds to a path of the same length and same weight in G;
P2. If a maximum k-path in G becomes a (directed) path in $\hat{G}_\pi$, then every maximum k-path in the directed acyclic graph $\hat{G}_\pi$ corresponds to a maximum k-path in the undirected graph G.

Now let $P_k = (w_1, w_1, \ldots, w_k)$ be a maximum k-path in the undirected graph G. We say that the path P_k *follows the order* of a permutation $\pi = (v'_1, v'_2, \ldots, v'_n)$ of $\{v_1, v_2, \ldots, v_n\}$ if for every i, w_{i+1} appears after w_i in the sequence $(v'_1, v'_2, \ldots, v'_n)$, and we say that the path P_k follows the *reversed order* of π if for every i, w_{i+1} appears before w_i in the sequence π. It is easy to see that if the maximum k-path P_k follows the order or the reversed order of the permutation π, then by Properties P1-P1, a maximum k-path in the directed acyclic graph $\hat{G}_\pi$ corresponds to a maximum k-path in the undirected graph G. Now a maximum k-path in the directed acyclic graph $\hat{G}_\pi$ can be constructed efficiently by the algorithm DagPath!

Therefore, the problem now is reduced to the problem of finding a good permutation of the vertices in the graph G. If we randomly pick a permutation of the vertices in G, what is the probability that the permutation is good?

Consider the sequence of n positions. Fix k positions of them, which will be the positions for the vertices of the k-path P_k to be placed in order or in the reversed order. There are $(n-k)!$ different permutations for the $n-k$ vertices that are not on the path P_k. Therefore, for these k positions, there are $2(n-k)!$ permutations of $\{v_1, v_2, \ldots, v_n\}$ for which the path P_k follows the order or the reversed order. Since there are $\binom{n}{k}$ ways to pick k positions out of the n positions, we conclude that the total number of permutations for which the path P_k follows the order or the reversed order is equal to

$$\binom{n}{k}(n-k)! = \frac{n!}{k!}.$$

Therefore, among the $n!$ permutations of $\{v_1, v_2, \ldots, v_n\}$, $n!/k!$ are good from which we can construct a maximum k-path of the undirected graph G using the algorithm DagPath. Thus, if we randomly pick a permutation of $\{v_1, v_2, \ldots, v_n\}$, then with a probability $(n!/k!)/n! = 1/k!$, we will pick a good permutation. This gives the following randomized algorithm for the max-Path problem:

Algorithm 10 PathPerm
Input: an undirected graph G, and an integer k;
Output: a maximum k-path in G;

1. **repeat** $t \cdot k!$ times
1.1 randomly pick a permutation π of $\{v_1, v_2, \ldots, v_n\}$;
1.2 construct the graph $\hat{G}_\pi$;
1.3 call DagPath to construct a maximum k-path in $\hat{G}_\pi$;
2. **if** no call in step 1.3 returns a k-path
2.1 **then** return("no k-path");
2.2 **else** return the k-path with the largest weight among those constructed in step 1.3.

First note that if the undirected graph G has no k-path, then no permutation π can make the directed acyclic graph $\hat{G}_\pi$ to have a k-path. Therefore, in this case, the algorithm PathPerm always returns at step 2.1 with a correct conclusion. On the other hand, if G has k-paths, then we fix any maximum k-path P_k in G. As we discussed above, a random permutation π has a probability $1/k!$ to have P_k follow the order or the reversed order of π. For such a permutation π, step 1.3 of the algorithm, which applies DagPath on the directed acyclic graph $\hat{G}_\pi$, will return a maximum k-path of $\hat{G}_\pi$ that, by properties P1-P2, corresponds to a maximum k-path of the undirected graph G. The probability that none of the $t \cdot k!$ random permutations picked in step 1.1 is good is bounded by

$$\left(1 - \frac{1}{k!}\right)^{t \cdot k!} < \frac{1}{e^t}.$$

In particular, if we let $t = 10$, then the algorithm PathPerm solves the max-Path problem with a probability at least 0.99999.

For any fixed constant t (e.g., $t = 10$), the running time of the randomized algorithm PathPerm is $O(k(n+m)k!)$, which is acceptable when the value of k is small.

Before we end this subsection, we illustrate how step 1.1 of the algorithm PathPerm picks a random permutation of $\{v_1, v_2, \ldots, v_n\}$. Presumably, each permutation of $\{v_1, v_2, \ldots, v_n\}$ should have an equal probability $1/n!$ to be picked. This can be implemented as followings. We simply pick a first vertex among all n vertices of the graph G, then pick a vertex from the remaining $n-1$ vertices, and then pick a vertex from the remaining $n-2$ vertices, and so on. Let $\pi = (v'_1, v'_2, \ldots, v'_n)$ be any given permutation of $\{v_1, v_2, \ldots, v_n\}$. We study the probability that this particular permutation π is picked. For each i, $1 \le i \le n$, let E_i be the event that the i-th picked vertex is v'_i, then the probability that this particular permutation π is picked is equal to

$$\begin{aligned}
& \Pr\left[\bigcap_{i=1}^{n} E_i\right] \\
= \; & \Pr\left[E_n \;\middle|\; \bigcap_{i=1}^{n-1} E_i\right] \cdot \Pr\left[E_{n-1} \;\middle|\; \bigcap_{i=1}^{n-2} E_i\right] \cdot \cdots\cdots \Pr\left[E_h \;\middle|\; \bigcap_{i=1}^{h-1} E_i\right] \cdot \cdots\cdots \Pr[E_2 \mid E_1] \cdot \Pr[E_1] \\
= \; & 1 \cdot \frac{1}{2} \cdot \cdots\cdots \frac{1}{n-h+1} \cdot \cdots\cdots \frac{1}{n-1} \cdot \frac{1}{n} \\
= \; & \frac{1}{n!},
\end{aligned}$$

where we have used the simple fact that if the i-th picked vertex is v'_i, for $1 \le i \le h-1$, then the probability that v'_h is the h-th picked vertex among the remaining vertices $\{v'_h, \ldots, v'_n\}$ is $1/(n-h+1)$, i.e., $\Pr[E_h \mid \bigcap_{i=1}^{h-1} E_i] = 1/(n-h+1)$. Thus, for each permutation π of $\{v_1, v_2, \ldots, v_n\}$, this process picks π with a probability $1/n!$, thus, it has an equal probability to pick any permutation of $\{v_1, v_2, \ldots, v_n\}$.

6.2 Color-coding

Color-coding is a technique proposed by Alon, Yuster, and Zwick [1]. The technique has proven very useful in dealing with problems in which one is interested in effectively finding a small subset with

certain desired properties from a large pool. The general principle of color-coding is similar to that of the method of random permutation as we described in the previous subsection for MAX-PATH: we first apply randomized methods that, with a sufficient probability, assign a special structure to a given problem instance so that the desired subset, which is hidden in the instance and unknown to the algorithm, can be identified more easily based on the special structure, then by taking advantage of the special structure, we apply (deterministic) techniques such as dynamical programming to identify the desired subset of our interests.

We will again use the MAX-PATH as an example. Let G be an undirected and weighted graph, whose vertex set is $V = \{v_1, v_2, \ldots, v_n\}$. We first examine why finding a maximum k-path is difficult when we apply a natural searching scheme. Suppose that we start from the vertex w_1, and try to find a maximum k-path starting from w_1. Assume somehow we have successfully extended the path from w_1 to the h-th vertex w_h in the path, $h < k$ (this is certainly good if $h = 1$). Because we are looking for simple paths, for further extending the path from w_h, we need to answer the following critical question:

What vertices have been used in the partial path from w_1 to w_h?

We could record the vertices of the maximum h-path from w_1 to w_h. However, since there can be more than one such maximum h-paths, recording a single such maximum h-path may not be good: if we recorded a wrong maximum h-path from w_1 to w_h containing some vertices that are necessary for further extending the path from w_h to w_k to get a maximum k-path, then we would not be able to get a maximum k-path. Thus, we seem to have to record *all* maximum h-paths from w_1 to w_h and consider a possible extension from each of them. Observing that for two such maximum h-paths that use the same set of h vertices, we only have to record one since we are only interested in which vertices have been used and do not care where they are used in the partial path, we can save some space: we only need to record each of those subsets of h vertices that makes a maximum h-path from w_1 to w_h.

Since there are $\binom{n}{h}$ subsets of h vertices in the graph G, where $\binom{n}{h} \approx n^h$ when h is significantly smaller than n, this would use a large amount of space (even when h is equal to, say, 10). Moreover, when we consider extending the path from the vertex w_h, we may need to look at each of these subsets, taking time at least $\Omega(n^h)$, also unacceptable.

The coloring-coding technique proposed an idea to take off this obstacle. For this, we first use k colors to color the vertices of the graph G, by which we mean a function f (a *k-coloring*) that maps the vertices of G to a set $C_k = \{c_1, c_2, \ldots, c_k\}$ of k colors, where $f(v_i)$ is called the *color* of the vertex v_i (under the k-coloring f). We say that a path P in G is *properly colored* if no two vertices in P are colored with the same color (of course, this implies that the path P contains at most k vertices).

Let f be a k-coloring of the vertices of G such that there is a maximum k-path $P_k = (w_1, w_2, \ldots, w_k)$ in G that is properly colored. Let us reconsider the above searching scheme and suppose our goal is to find the path P_k. Again assume that we have successfully extended from w_1 to the h-th vertex w_h in the path P_k. Observe that now only the colors used on the partial path of P_k from w_1 to w_h need to be recorded: since P_k is properly colored, only those vertices whose colors have not been used in the partial path from w_1 to w_h need to be considered. Therefore, instead of recording the subsets of h vertices in the set of n vertices in the graph G that can make maximum h-paths from w_1 to w_h (there can be up to $\binom{n}{h}$ of them), now we only have to record at most $\binom{k}{h}$ subsets of h colors in the set C_k of k colors that can make properly colored h-paths of maximum weight from w_1 to w_h. The number $\binom{k}{h}$ is significantly smaller than $\binom{n}{h}$ when k is small compared with n, thus will significantly save the space and time in the above searching scheme.

This idea is implemented in the algorithm COLORPATH given below that on a k-colored graph G finds a properly colored k-path of the maximum weight. For each vertex v_i in G, we keep a collection $A[v_i]$ of triples of the form $(S, \omega, x)_i$ (where the subscript of the parentheses indicates the vertex to which the triple belongs). Each triple $(S, \omega, x)_i$ in $A[v_i]$, where S is subset of the color set C_k, records a properly colored path P that satisfies the following conditions: (1) P ends at the vertex v_i; (2) P uses exactly those colors in S (thus, the color $f(v_i)$ of v_i is in S, and P is a $|S|$-path); (3) the weight of P is ω; and (4) x is the vertex just before v_i on the path P. Naturally, the triple $(\{f(v_i)\}, 0, *)_i$ is contained in the collection $A[v_i]$, representing the 1-path P_1 starting and ending at the vertex v_i. Since the path P_1 has no vertex before v_i, the third component of the triple is marked by an irrelevant symbol $*$.

Algorithm 11 ColorPath
Input: an undirected weighted graph G that is k-colored, and an integer k;
Output: a properly colored k-path of maximum weight in G;

1. **for** each vertex v_i of G **do** $A[v_i] = \{\ (\{f(v_i)\}, 0, *)_i\ \}$;
2. **for** $h = 1$ **to** $k-1$ **do**
   for each vertex v_i of G **do**
   for each edge $[v_i, v_j]$ in G **do**
   for each $(S, \omega, x)_i$ in $A[v_i]$, where S is an h-subset of C_k and $f(v_j) \notin S$ **do**
   if $(S \cup \{f(v_j)\}, \omega', x')_j$ is not in $A[v_j]$ or
   $(S \cup \{f(v_j)\}, \omega', x')_j$ in $A[v_j]$ satisfies $\omega' < \omega + \mathrm{wt}(v_i, v_j)$
   then change $(S \cup \{f(v_j)\}, \omega', x')_j$ in $A[v_j]$ to $(S \cup \{f(v_j)\}, \omega'', v_i)_j$,
   where $\omega'' = \omega + \mathrm{wt}(v_i, v_j)$;
3. The vertex v_t such that the triple $(C_k, \omega, *)_t$ in $A[v_t]$ has the largest ω is the last vertex of a properly colored k-path of maximum weight.

The correctness of the algorithm ColorPath is given by the following lemma.

Lemma 6.1 *Let $P_k = (w_1, w_2, \ldots, w_k)$ be a properly colored k-path of the maximum weight in the k-colored graph G. For each h, $0 \le h \le k-1$, after the h-th execution of the h-loop in step 2 of the algorithm* ColorPath*, the collection $A[w_{h+1}]$ for the vertex w_{h+1} contains a triple for a properly colored $(h+1)$-path that uses the colors $f(w_1)$, $f(w_2)$, ..., $f(w_{h+1})$ and has weight equal to that of the partial path $(w_1, w_2, \ldots, w_{h+1})$.*

proof. The lemma can be proved by induction on h. For $h = 0$, the lemma is ensured because step 1 of the algorithm. Now consider the case for $h > 0$. By the inductive hypothesis, after the $(h-1)$-st execution of the loop of step 2 in the algorithm, the collection $A[w_h]$ for the vertex w_h contains a triple $(S, \omega, x)_h$ for a properly colored h-path P_h such that $S = \{f(w_1), f(w_2), \ldots, f(w_h)\}$, and ω is equal to the weight of the partial path $(w_1, w_2, \ldots, w_h)$.

During the h-th execution, when the edge $[w_h, w_{h+1}]$ and the trip $(S, \omega, x)_h$ in $A[w_h]$ are examined, if the triple $(S \cup \{f(w_{h+1})\}, \omega', x')_j$ does not exist in $A[w_{h+1}]$ or it exists but $\omega' < \omega + \mathrm{wt}(w_h, w_{h+1})$, then by the algorithm, the collection $A[w_{h+1}]$ will contain a triple $(S', \omega'', v_i)_{h+1}$, where $S' = S \cup \{f(w_{h+1})\} = \{f(w_1), \ldots, f(w_h), f(w_{h+1})\}$, $\omega'' = \omega + \mathrm{wt}(w_h, w_{h+1}) =$ the weight of the partial path $(w_1, w_2, \ldots, w_{h+1})$, giving the $(h+1)$-path that is the concatenation of the h-path given by the triple $(S, \omega, x)_h$ in $A[w_h]$ and the edge $[w_h, w_{h+1}]$. This completes the induction.

Note that a triple (S, ω, x) is added to the collection $A[v_i]$ for a vertex v_i only if we have seen the corresponding path. Thus, the collection $A[w_h]$ cannot contain a triple $(\{f(w_1), f(w_2), \ldots, f(w_h)\}, \omega, x)$ where ω is larger than the weight of the h-path $(w_1, w_2, \ldots, w_h)$ – otherwise, there would be a properly colored k-path whose weight is larger than that of P_k. Therefore, once a triple corresponding to the path $(w_1, w_2, \ldots, w_h)$ is established in the collection $A[w_h]$ for the vertex w_h, it will stay there forever.

This completes the proof of the lemma. □

By Lemma 6.1, after step 2, some vertices (in particular vertex w_k) will have their collection containing a triple that gives a properly colored k-path whose weight is equal to that of P_k, i.e., a properly colored k-path of the maximum weight. Step 3 of the algorithm will return such a path. Also note that the returned path can be re-constructed based on the third component of the triples.

Since the collection $A[v_i]$ for a vertex v_i contains at most 2^k triples, corresponding to the 2^k subsets of the color set C_k, it is easy to see that the algorithm ColorPath runs in time $O(2^k k(n+m))$. This is a significant improvement over the running time $O(k(n+m)k!)$ of the algorithm PathPerm that is based on random permutation.

However, we are not done, yet. The graph given by the problem max-Path is not colored. How do we color the graph so that a maximum k-path in the graph is properly colored?

Again we make use of randomization. So our question is: if we randomly color the n vertices of a given undirected and weighted graph G using k colors, what is the probability that a maximum k-path in G is properly colored?

By the fundamental counting principle, there are total k^n different ways to k-color the n vertices of the graph G. Let $P_k = (w_1, w_2, \ldots, w_k)$ is a fixed maximum k-path in G. There are $k!$ k-colorings that properly color the k vertices in P_k. For each proper k-coloring of the k vertices in P_k, there are k^{n-k} ways to k-color the $n-k$ vertices that are not in P_k. Therefore, indeed, there are totally $k^{n-k}k!$ different k-colorings that properly color the path P_k. In summary, the ratio of the number of k-colorings that properly color the path P_k over the total number of k-colorings is

$$\frac{k^{n-k}k!}{k^n} = \frac{k!}{k^k} \geq \frac{\sqrt{2\pi k}(k/e)^k}{k^k} = \frac{\sqrt{2\pi k}}{e^k} > \frac{1}{e^k},$$

here we have used Stirling's formula (Theorem 5.1), and $e = 2.718\cdots$ is the base of the natural logarithm. Therefore, if we randomly color the vertices of the graph G, we will have a probability at least $1/e^k$ to color the maximum k-path P_k properly.

Now we are ready for the final algorithm for the problem MAX-PATH based on color-coding.

Algorithm 12 PATHCC
Input: an undirected and weighted graph G, and an integer k;
Output: a maximum k-path in G;

1. **repeat** te^k times
   randomly k-color the vertices of the graph G, let the k-colored graph be G';
   call COLORPATH on G' to construct a properly colored k-path of maximum weight;
2. **if** no call above to COLORPATH returns a k-path
   then return("no k-path");
   else return the k-path with the largest weight among those constructed in step 1.

Based on analysis similar to that given for the algorithm PATHPERM in the previous subsection, we have the following remarks:

- If the graph G has no k-path, then the algorithm PATHCC always reports correctly;
- If the graph G contains k-paths, then with a probability at least $1-(1-1/e^k)^{te^k} > 1-1/e^t$, the algorithm returns a maximum k-path of the graph G. In particular, if we set $t = 10$, then the algorithm has a success probabablity larger than 0.99999.
- Since the algorithm COLORPATH runs in time $O(2^k k(n+m))$, the time complexity of the algorithm PATHCC is $O(t(2e)^k k(n+m))$, which is bounded by $O(5.44^k(n+m))$ for any constant t, which is an improvement over the algorithm PATHPERM, which is based on random permutation and has running time $O(k(n+m)k!)$: for $k \geq 15$, $k! \geq 5.44^k$.
- Step 1 for uniformly picking a k-coloring of the graph G over all k-colorings for G can be implemented by first picking any color from the k colors, with an equal probability, for the first vertex, then picking a color from the k colors with an equal probability for the second vertex, and so on.