Lecture Outline (Week 10)
Topics in Graph Algorithms (CSCI5320-20S)

CAI Leizhen
Department of Computer Science and Engineering
The Chinese University of Hong Kong
lcai@cse.cuhk.edu.hk

April 8, 2020

Keywords: Random partition, and random separation.

1 Random Partition

Basic idea: Randomly partition an instance into two parts, and then independently solve problems for the two parts. We can use universal sets to derandomize such algorithms.

1. (n,t)**-universal sets**: A collection of binary vectors of length n is (n,t)-*universal* if for every subset of size t of the indices, all 2^t configurations appear. Naor, Schulman and Srinivasan have a construction for (n,t)-universal sets of size $2^t t^{O(\log t)} \log n$ that can be listed in time $2^t t^{O(\log t)} n \log n$.

2. Disjoint Paths

   Task: Find two vertex-disjoint k-paths P_1 and P_2 in graph G.

   Step 1. Randomly partition vertices of G into V_1 and V_2 to form graphs $G_1 = G[V_1]$ and $G_2 = G[V_2]$.

   Step 2. Find a k-path P_1 in G_1 and a k-path P_2 in G_2.

   Let $T(m,n)$ be the time for finding a k-path in a graph with m edges and n vertices. When G admits a solution, the above algorithm finds a solution in time $T(m,n)$ with probability 2^{-2k}, as each P_i $(i = 1,2)$ has probability 2^{-k} to be entirely inside graph G_i. We can use a family of $(n,2k)$-universal sets of size $2^{2k}k^{O(\log k)} \log n$ to derandomize the algorithm and obtain a deterministic algorithm with running time

$$2^{2k}k^{O(\log k)} \log n T(m,n) = 4^{k+O(\log^2 k)} T(m,n) \log n = O(4.01^k T(m,n) \log n).$$

3. Disjoint Paths: one short and one unconstrained

   L. Cai and J. Ye, Finding Two Edge-Disjoint Paths with Length Constraints, WG 2016, LNCS 9941 pp. 62-73, 2016.

   Task: For a pair (s, t) of vertices in a graph G, find edge-disjoint (s, t)-paths P and Q such that P has length at most k.

   Definition 1.1 *A vertex v is a* nearby-vertex *if* $\min\{d(v, s), d(v, t)\} \leq k/2$*, and an edge is a* nearby-edge *if its two endpoints are both nearby-vertices.*

   The following lemma is a key for an FPT algorithm based on random partition.

   Lemma 1.2 *Let (s, t) be a pair of vertices in a graph $G = (V, E)$, P an (s, t)-path of length at most k, and Q a minimum-length (s, t)-path edge-disjoint from P. Then*

   (a) all edges in P are nearby-edges, and

   (b) Q contains at most $(k + 1)^2 - 1$ nearby-edges.

   Proof. Statement 1 is obvious and we focus on Statement 2. For this purpose, we call a vertex a *P-near vertex* if its distance to P is at most $k/2$, and we first give an upper bound on the number of P-near vertices in Q. Consider an arbitrary vertex x in P, and define

   $$N_x^* = \{v : v \text{ is a nearby-vertex in } Q \text{ and } d(v, x) = d(v, P)\},$$

   where $d(v, P)$ is the minimum distance between v and any vertex of P. In other words, for each vertex $v \in N_x^*$, x is a vertex in P closest to v.

   Order vertices in N_x^* along Q from s to t, and let x_s and x_t be the first and last vertices, respectively. Let P_s be a shortest (x_s, x)-path and P_t a shortest (x, x_t)-path in G. Then both P_s and P_t are edge-disjoint from P as x is a vertex in P closest to both x_s and x_t, and therefore $P_s P_t$ is an (x_s, x_t)-walk edge-disjoint from P.

   Note that $P_s P_t$ contains at most k edges as both P_s and P_t have at most $k/2$ edges. If the (x_s, x_t)-section of Q contains more than k edges, then we can replace it by $P_s P_t$ to obtain an (s, t)-walk that is edge-disjoint from P and shorter than Q, contradicting the minimality of Q. Therefore, the (x_s, x_t)-section of Q contains at most k edges, implying that it contains at most $k + 1$ P-near vertices, i.e., $|N_x^*| \leq k + 1$.

   Since P has at most $k + 1$ vertices, and every P-near vertex in Q belongs to N_x^* for some vertex x in P, we see that Q contains at most $(k + 1)^2$ P-near vertices. From the definition of nearby-vertices, we know that every nearby-vertex is a P-near vertex as s and t are vertices of P. Therefore Q contains at most $(k + 1)^2$ nearby-vertices, and hence at most $(k + 1)^2 - 1$ nearby-edges. ∎

Let $\{E_1, E_2\}$ be a random partition of nearby-edges, and construct $G_1 = G[E_1]$ and $G_2 = G - E(G_1)$. Note that whenever G admits a solution, it has a solution (P, Q) such that Q is a minimum-length (s,t)-path edge disjoint from P. Lemma 1.2 implies that P is inside G_1 with probability $\geq 1/2^k$, and Q is inside G_2 with probability $\geq 1/2^{(k+1)^2}$. This ensures that, with probability $\geq 1/2^k$, G_1 contains an (s,t)-path of length at most k and, with probability at least $1/2^{(k+1)^2}$, G_2 contains an (s,t)-path. Therefore with probability $\geq 1/2^{k+(k+1)^2}$, we will be able to find a solution for G by finding an (s,t)-path of length at most k in G_1 and an (s,t)-path in G_2.

Algorithm DP1S:

(a) Find all nearby-edges in $O(m)$ time by two rounds of BFS, one from s and the other from t.

(b) Randomly color each nearby-edge by color 1 or 2 with probability 1/2, and color all remaining edges of G by color 2. Let G_i $(i = 1, 2)$ be the graph consisting of edges of color i.

(c) Find an (s,t)-path P of length $\leq k$ in G_1, and an (s,t)-path Q in G_2. Return (P, Q) as a solution if both P and Q exist, and return "No" otherwise.

Algorithm DP1S solves our problem with probability $\geq 1/2^{k+(k+1)^2}$ and runs in $O(m)$ time, as the two tasks in Step (c) for G_1 and G_2 also take $O(m)$ time. Let m' be the number of nearby-edges and $r = k + (k+1)^2$. We can use (m', r)-universal sets to derandomize our algorithm, and obtain a deterministic FPT algorithm running in time

$$2^r r^{O(\log r)} \log n * m' = O(2.01^{k^2} m \log n).$$

2 Random Separation (Cai, Chan and Chan 2006)

L. Cai, S.M. Chan and S.O. Chan, Random separation: a new method for solving fixed-cardinality optimization problems, LNCS 4169 (pp.239-250), 2006.

The basic idea of this innovative method is to use a random partition of the vertex set V of a graph $G = (V, E)$ to separate a solution from the rest of G into connected components and then select appropriate components to form a solution. Algorithms obtained from this method can be derandomized by families of universal sets.

Random separation is very effective for a large variety of parameterized problems on graphs with bounded degree or bounded degeneracy, and also useful for some parameterized problems on general graphs.

1. Dense k-Vertex Subgraphs (a.k.a. Maximum k-Vertex Subgraph) for degree-bounded graphs (Cai, Chan and Chan 2006)

   Let $G = (V, E)$ be a graph of maximum degree d for some constant d. Find k vertices V' in G to maximize the number of edges in $G[V']$.

   First we randomly colour each vertex of G by either green or red each with probability $1/2$ to form a random partition (V_g, V_r) of V. Green vertices V_g induce the *green subgraph* $G_g = G[V_g]$, and the connected components of G_g are *green components*.

   Let G' be a maximum k-vertex induced subgraph of G. A random partition of V is a "good partition" for G' if all vertices in G' are green and all vertices in its neighbourhood $N_G(G')$ are red. Note that $N_G(G')$ has at most dk vertices as $d_G(v) \leq d$ for each vertex v. Therefore the probability that a random partition is a good partition for G' is at least $2^{-(d+1)k}$ and thus, with at least this probability, G' is the union of some green components.

   To find a maximum k-vertex induced subgraph for a good partition of G', we need only find a collection $\mathcal{H}'$ of green components such that the total number of vertices in $\mathcal{H}'$ is k and the total number of edges in $\mathcal{H}'$ is maximized. For this purpose, we first compute in $O(dn)$ time the number n_i of vertices and the number m_i of edges inside each green component H_i. Then we find a collection $\mathcal{H}'$ of green components that maximizes

   $$\sum_{H_i \in \mathcal{H}'} m_i$$

   subject to $\sum_{H_i \in \mathcal{H}'} n_i = k$.

   Since for any two green components H_i and H_j, the number of vertices (resp. the number of edges) in $H_i \cup H_j$ equals $n_i + n_j$ (resp. $m_i + m_j$), we can solve the problem in $O(kn)$ time by the standard dynamic programming algorithm for the 0-1 Knapsack problem. Therefore, with probability at least $2^{-(d+1)k}$, we can find a maximum k-vertex induced subgraph of G in $O((d + k)n)$ time.

   To derandomize the algorithm, we use a family of partitions with the property that for every partition Π of any $(d + 1)k$ vertices into k vertices and dk vertices, there is a partition in the family that is consistent with Π.

   We can interpret a binary vector of length n as a red-green coloring of G: vertex v_i is colored green if the i-th position of the vector is 1 and red if 0. Then a family of $(n, (d + 1)k)$-universal sets can be used as the required family of partitions. Therefore we obtain an FPT-algorithm that runs in $O(f(k, d)n \log n)$ time where

   $$f(k, d) = 2^{(d+1)k}(dk + k)^{O(\log(dk+k))}(d + k).$$