Counting s–t paths of at most length ℓ via DLX

Satoshi Matsumoto *[1], Motohiro Higa[1], Takashi Harada[1], and Shogo Takeuchi[1]

[1] *School of Informatics, Kochi University of Technology*

October 16, 2024

Abstract

In this paper, we show an algorithm counting s–t paths of length at most ℓ for an undirected graph. Our algorithm comprises of three parts: (1) preprocessing an input graph, (2) generating an instance of exact coverring with colors problem (XCC), and (3) solving it via Dancing Links.

1 Problem

The problem in the category of Single-thread solvers for undirected graphs in ICGCA 2024 is described below:

Input: An undirected graph $G = (V, E)$, terminal nodes s, t, and the maximum path length ℓ.

Output: The number of simple paths between s and t of length at most ℓ.

This problem is known as #P-complete [1].

2 Approach

Our algorithm consists of the following three parts:

1. Preprocessing an input graph.
   (a) Removing redundant nodes.
   (b) Converting the unweighted graph to the edge weighted graph.
2. Generating an intance of XCC for the given input.
3. Solving it via Dancing Links.

2.1 Removing redundant nodes

In order to remove redundant nodes, we use biconnected components. A biconnected component of G is a maximal edge set where any two edges in the set are on a common simple cycle. An articulation point of G is a node which become disjoint when it is deleted. They can obtained in linear time to the size of the edge set by Depth-first search [2].

First, we compute the biconnected components and articulation points of the input graph $G = (V, E)$ by using Depth-first search. For each biconnected component G_B that does not contain s and t, we select a node v_x in G_B except an articulation point. Let P be a shortest path from s to v_x and G' denote a graph $G \setminus (P \setminus \{v_x\})$. If there is no path from v_x to t on G', we remove edges on G_B from G, because a path containing an edge in G_B is not a simple path from s to t.

Furthermore, we remove leaf nodes from the graph.

*Corresponding author (250371g@ugs.kochi-tech.ac.jp)

2.2 Converting to the weighted graph

We firstly convert the undirected graph that has no redundant nodes to an weighted graph whose edges have weight 1. Let U be a set of nodes whose degree is 2. For each node $v \in U \setminus \{s, t\}$, there are two nodes u and v such that $\{w, v\} \in E$ and $\{v, u\} \in E$. We remove them from E and add the edge $\{w, u\}$ with weight α to E, where α is the sum of weights of $\{w, v\}$ and $\{v, u\}$.

2.3 Generating an instance of XCC

Exact cover problem (XC) is the following problem:

Input: A set U and a subset S of the power set of U.

Output: A subset S' of S such that S' is a partition of U.

In the following, we call U an item set and S an option set. We divide a item set into a set of primary items and a set of secondary items. A primary item must be covered exactly once as original exact cover problem, and a secondary item must be covered at most once.

Furthermore, a secondary item can have at most one color. We denote a secondary item having a color by a colon; for example '`p:A`' means that color `A` is assigned to the secondary item `p`. If two option have the same secondary item, you can take these options when those colors are the same. Exact covering with colors (XCC) introduced by Knuth [3] is the following problem: An input is a set of items and a set of options. In contrast to the original XC, a color is assigned to a secondary item of each option. An output is a set of options such that

1. every primary item is covered exactly once; and
2. every secondary item is assigned at most one color.

Let #1, #2, ..., #ℓ, and T be primary items, and v_1, v_2, ..., v_n, p_0, p_1, ..., p_ℓ be secondary items, where n is the number of nodes. For an input graph, our algorithm generates options by the following processes:

1. For each node v, it computes the lengths of shortest paths ls_v and lt_v from the terminal nodes s and t.
2. For each edge $\{x, y\}$ with weight c, for all $k \geq 0$, our algorithm generates the following option
$$\{\#A, \#(A+1), \ldots, \#B, v_x : A, v_y : B, p_A : v_x, p_B : v_y\},$$
where $A = ls_x + k$ and $B = A + c$. An option also includes the primary item T when $y = t$.
3. For all $lt_s \leq k < \ell$, it generates the option $\{\#k, \#(k+1), \ldots, \#\ell\}$.

Each option means that an edge $\{x, y\}$ is selected at $ls_x + k$. Options consisting only of primary items are needed for a paths of length not exactly ℓ but less than ℓ.

2.4 Solving by XCC solver

The generated instance as input to an XCC solver, and compute the number of solutions. Our solver is based on Algorithm C [3] using Dancing Links.

References

[1] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, Vol. 8, No. 3, pp.410–421, 1979

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stain. *Introduction to Algorithms, Fourth Edition.*, The MIT Press, 2022.

[3] Donald E. Knuth. *The Art of Computer Programming Volume 4B: Combinatorial Algorithms, Part 2.* Addison-Wesley Professional, 2022.

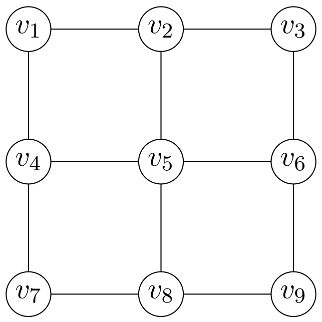

Figure 1: $s = v_1, t = v_9$, and $\ell = 8$.

A Example of instance

The instances for the graphs shown in Fig.1 are as follows. The first line shows items. The symbol '|' separates the primary items and secondary items. The remaining lines show options.

```
#1 #2 #3 #4 #5 #6 #7 #8 t | v1 v2 v3 v4 v5 v6 v7 v8 v9 p0 p1 p2 p3 p4 p5 p6 p7 p8
#1 v1:0 v2:1 p0:1 p1:2
#1 v1:0 v4:1 p0:1 p1:4
#2 v2:1 v5:2 p1:2 p2:5
#3 v2:2 v5:3 p2:2 p3:5
#4 v2:3 v5:4 p3:2 p4:5
#5 v2:4 v5:5 p4:2 p5:5
#6 v2:5 v5:6 p5:2 p6:5
#2 #3 v2:1 v6:3 p1:2 p3:6
#3 #4 v2:2 v6:4 p2:2 p4:6
#4 #5 v2:3 v6:5 p3:2 p5:6
#5 #6 v2:4 v6:6 p4:2 p6:6
#6 #7 v2:5 v6:7 p5:2 p7:6
#2 v4:1 v5:2 p1:4 p2:5
#3 v4:2 v5:3 p2:4 p3:5
#4 v4:3 v5:4 p3:4 p4:5
#5 v4:4 v5:5 p4:4 p5:5
#6 v4:5 v5:6 p5:4 p6:5
#2 #3 v4:1 v8:3 p1:4 p3:8
#3 #4 v4:2 v8:4 p2:4 p4:8
#4 #5 v4:3 v8:5 p3:4 p5:8
#5 #6 v4:4 v8:6 p4:4 p6:8
#6 #7 v4:5 v8:7 p5:4 p7:8
#3 v5:2 v2:3 p2:5 p3:2
#4 v5:3 v2:4 p3:5 p4:2
#5 v5:4 v2:5 p4:5 p5:2
#3 v5:2 v4:3 p2:5 p3:4
#4 v5:3 v4:4 p3:5 p4:4
#5 v5:4 v4:5 p4:5 p5:4
#3 v5:2 v6:3 p2:5 p3:6
#4 v5:3 v6:4 p3:5 p4:6
#5 v5:4 v6:5 p4:5 p5:6
#6 v5:5 v6:6 p5:5 p6:6
#7 v5:6 v6:7 p6:5 p7:6
#3 v5:2 v8:3 p2:5 p3:8
#4 v5:3 v8:4 p3:5 p4:8
#5 v5:4 v8:5 p4:5 p5:8
#6 v5:5 v8:6 p5:5 p6:8
#7 v5:6 v8:7 p6:5 p7:8
#4 #5 v6:3 v2:5 p3:6 p5:2
#4 v6:3 v5:4 p3:6 p4:5
#5 v6:4 v5:5 p4:6 p5:5
```

```
#6 v6:5 v5:6 p5:6 p6:5
#4 v6:3 v9:4 p3:6 p4:9 t
#5 v6:4 v9:5 p4:6 p5:9 t
#6 v6:5 v9:6 p5:6 p6:9 t
#7 v6:6 v9:7 p6:6 p7:9 t
#8 v6:7 v9:8 p7:6 p8:9 t
#4 #5 v8:3 v4:5 p3:8 p5:4
#4 v8:3 v5:4 p3:8 p4:5
#5 v8:4 v5:5 p4:8 p5:5
#6 v8:5 v5:6 p5:8 p6:5
#4 v8:3 v9:4 p3:8 p4:9 t
#5 v8:4 v9:5 p4:8 p5:9 t
#6 v8:5 v9:6 p5:8 p6:9 t
#7 v8:6 v9:7 p6:8 p7:9 t
#8 v8:7 v9:8 p7:8 p8:9 t
#5 #6 #7 #8
#6 #7 #8
#7 #8
#8
```