Submission for International Competition on Graph Counting Algorithms

Shou Ohba*

July 4, 2023

*Kyoto University

1 Introduction

We implemented a portfolio solver which uses the following algorithms.

1. Depth-First Search
2. Meet in the middle
3. Simpath[1]
   - 3-a. default edge ordering
   - 3-b. community-based edge ordering

The criteria for the use are as follows.

- For the instances whose maximum path length l is less than or equal to 14, we run the algorithm 2.
- For the other instances, we run the algorithms 1, 3-a, and 3-b in parallel.

2 Depth-First Search

For instances that have few paths, a straightforward depth-first search (DFS) is effective enough.

For all-pairs instances, for each vertex s, we run DFS starting from s and enumerate simple paths. To avoid double-counting, we count paths whose terminals are not less than s. Computations can be performed independently for each s, so we can parallelize this algorithm easily.

We can improve this algorithm if the input graph has small neighborhood diversity[2]. To introduce neighborhood diversity, we define the relation $\sim_{\mathrm{nd}}$ on V as follows:

$$u \sim_{\mathrm{nd}} v \overset{\mathrm{def}}{\iff} N(u) \setminus \{v\} = N(v) \setminus \{u\},$$

where $N(v)$ denotes the neighborhood of $v \in V$. This is an equivalence relation in fact, and neighborhood diversity is defined as the size of the quotient set $V/{\sim_{\mathrm{nd}}}$.

We represent a not necessarily simple path P as the sequence of vertices $(P_0, \dots, P_k)$.

Let P be a simple path and assume that there exist integers i, j $(0 \le i < j \le k)$ which satisfy $P_i \sim_{\mathrm{nd}} P_j$. By swapping P_i and P_j, we obtain a sequence $P' = (P_0, \dots, P_{i-1}, P_j, P_{i+1}, \dots, P_{j-1}, P_i, P_{j+1}, \dots, P_k)$. Then P' is also a path because of the definition of $\sim_{\mathrm{nd}}$ and the simplicity of P. This means that vertices u and v are equivalent when counting simple paths if they satisfy $u \sim_{\mathrm{nd}} v$.

Now we construct a new graph $G'(V/{\sim_{\mathrm{nd}}}, E')$, where the edge set E' is defined as follows:

$$E' = \{\{[u],[v]\} \mid \exists a \in [u], \exists b \in [v] \text{ s.t. } \{a,b\} \in E\}.$$

Note that G' does not contain multi-edges, but may contain self-loops.

For a simple path $P = (P_0, \ldots, P_k)$ on G, we defines $[P] = ([P_0], \ldots, [P_k])$. $[P]$ is a not necessarily simple path on G'. For a not necessarily simple path Q on G', the number of simple paths P on G which satisfies $[P] = Q$ can be expressed as follows:

$$\prod_{[u]\in V/\sim_{\mathrm{nd}}} {}_{W_{[u]}}P_{C^Q_{[u]}} = \prod_{[u]\in V/\sim_{\mathrm{nd}}} \left(W_{[u]}(W_{[u]}-1)\cdots(W_{[u]}-C^Q_{[u]}+1) \right),$$

where $W_{[u]} = |[u]|$ and $C^Q_{[u]} = |\{i \mid Q_i = [u]\}|$. Similarly, the number of simple s–t paths P on G which satisfies $[P] = Q$ can be expressed as follows:

$$\begin{cases} 0 & \text{if } s \notin Q_0 \vee t \notin Q_k \\ {}_{W_{[s]}-2}P_{C^Q_{[s]}-2} \cdot \prod_{[u]\in(V/\sim_{\mathrm{nd}})\setminus\{[s]\}} {}_{W_{[u]}}P_{C^Q_{[u]}} & \text{if } s,t \in Q_0 = Q_k \\ {}_{W_{[s]}-1}P_{C^Q_{[s]}-1} \cdot {}_{W_{[t]}-1}P_{C^Q_{[t]}-1} \cdot \prod_{[u]\in(V/\sim_{\mathrm{nd}})\setminus\{[s],[t]\}} {}_{W_{[u]}}P_{C^Q_{[u]}} & \text{otherwise} \end{cases}$$

From the above, simple paths on G can be counted by counting not necessarily simple paths on G' with appropriate weights.

3 Meet in the middle

We split a simple path $P = (P_0, P_1, \ldots, P_k)$ by the midpoint $x = P_{\lceil k/2 \rceil}$. We enumerate the first half simple paths $(x = P_{\lceil k/2\rceil}, P_{\lceil k/2\rceil-1}, \ldots, P_0)$ and the second half simple paths $(x = P_{\lceil k/2\rceil}, P_{\lceil k/2\rceil+1}, \ldots, P_k)$ independently. Finally, we count the pairs of paths which form a simple path by connecting at x.

3.1 all-pairs

We fix the midpoint $x \in V$ and the length k $(1 \le k \le l)$. Here l is the maximum length of paths. Let $\mathcal{P}_{x,i}$ be the set of simple paths with length i which starts from $x \in V$.

We combine simple paths $L = (x = L_0, L_1, \ldots, L_{\lceil k/2 \rceil}) \in \mathcal{P}_{x,\lceil k/2 \rceil}$ and $R = (x = R_0, R_1, \ldots, R_{\lfloor k/2 \rfloor}) \in \mathcal{P}_{x,\lfloor k/2 \rfloor}$ to obtain a new path $P = (L_{\lceil k/2 \rceil}, L_{\lceil k/2 \rceil - 1}, \ldots, L_1, x, R_1, R_2, \ldots, R_{\lfloor k/2 \rfloor})$. P is simple if and only if $V(L) \cap V(R) = \{x\}$ holds. Here $V(S)$ denotes the set of vertices appearing in S.

For a path $L \in \mathcal{P}_{x,\lceil k/2 \rceil}$, let $C_{x,k}(L)$ be the number of paths $R \in \mathcal{P}_{x,\lfloor k/2 \rfloor}$ which satisfies $V(L) \cap V(R) = \{x\}$. From the inclusion-exclusion principle, $C_{x,k}(L)$ can be computed as follows:

$$C_{x,k}(L) = \sum_{S \subseteq (V(L) \setminus \{x\})} (-1)^{|S|} \cdot |\{R \in \mathcal{P}_{x,\lfloor k/2 \rfloor} \mid (V(R) \setminus \{x\}) \supseteq S\}|.$$

Computing $D_{x,k}(S) = |\{R \in \mathcal{P}_{x,\lfloor k/2 \rfloor} \mid (V(R) \setminus \{x\}) \supseteq S\}|$ naively is costly, so we precompute $D_{x,k}(S)$ for all $S \subseteq V$. This can be done in the following way:

1. Make an empty hash map $D_{x,k}$.
2. For each $R \in \mathcal{P}_{x,\lfloor k/2 \rfloor}$, do the following:
   (a) For each $S \subseteq V(R) \setminus \{x\}$, do the following:
      - If S is not contained in $D_{x,k}$ as a key, update $D_{x,k}(S)$ as $D_{x,k}(S) \leftarrow 1$.
      - Otherwise, update $D_{x,k}(S)$ as $D_{x,k}(S) \leftarrow D_{x,k}(S) + 1$.

This precomputation can be done in (expected) $O(|\mathcal{P}_{x,\lfloor k/2 \rfloor}| k 2^{\lfloor k/2 \rfloor})$ time. Using the precomputing result, we can calculate all $C_{x,k}(L)$ in (expected) $O(|\mathcal{P}_{x,\lceil k/2 \rceil}| k 2^{\lceil k/2 \rceil})$ time.

We must not distinguish between paths that differ only in direction. Thus, the answer can be written as $\displaystyle\frac{1}{2} \sum_{x \in V} \sum_{k=1}^{l} \sum_{L \in \mathcal{P}_{x,\lceil k/2 \rceil}} C_{x,k}(L)$.

Furthermore, from the discussion in the previous section, the answer can also be expressed as $\displaystyle\frac{1}{2} \sum_{[x] \in V/\sim_{\mathrm{nd}}} W_{[x]} \sum_{k=1}^{l} \sum_{L \in \mathcal{P}_{x,\lceil k/2 \rceil}} C_{x,k}(L)$.

Computations can be performed independently for each x, so we can parallelize this algorithm easily.

3.2 one-pair

We fix the midpoint $x \in V$ and the length of paths k $(1 \leq k \leq l)$. Let $\mathcal{L}_{x,i}$ and $\mathcal{R}_{x,i}$ be the set of simple $x{-}s$ and $x{-}t$ paths with length i, respectively.

By applying the inclusion-exclusion principle in the same way as for all-pairs instances, the answer is expressed as follows:

$$\sum_{x \in V} \sum_{k=1}^{l} \sum_{L \in \mathcal{L}_{x,\lceil k/2 \rceil}} \sum_{S \subseteq V(L) \setminus \{x\}} (-1)^{|S|} \cdot |\{R \in \mathcal{R}_{x,\lfloor k/2 \rfloor} \mid (V(R) \setminus \{x\}) \supseteq S\}|.$$

We can improve this algorithm for instances with small neighborhood diversity as well as for all-pairs instances.

4 Simpath[1]

Simpath is an algorithm to enumerate simple paths using Zero-suppressed Binary Decision Diagrams (ZDDs)[3]. We will not discuss Simpath in detail here but rather describe the ordering of edges.

4.1 default order

If the input is already organized, the default ordering can be effective enough.

4.2 community-based order

Networks are sometimes shaped in such a way that they sparsely connect between dense communities. Thus, we try the following edge ordering:

1. By applying Girvan-Newman algorithm[4], divide the vertex set to some communities. Girvan-Newman algorithm makes several candidate partitions, thus we need to choose one of them. This time we adopted the one with the highest modularity[5].

2. Relabel the vertices so that vertices in the same community have consecutive labels. Communities are ordered in preorder of a DFS tree of the graph obtained by contracting communities. For one-pair instances, we choose the community that contains the terminal s as a root of the DFS tree. For all-pairs instances, we choose at random.

3. Order edges $\{u, v\}$ in the ascending order of $(\min(u, v), \max(u, v))$.

For example, Figure 1 shows the input graph of `one-pair/052.col`. The first step outputs the division shown in Figure 2, and the second step outputs the labeling of vertices shown in Figure 3.

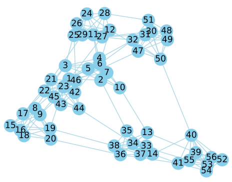

Figure 1: Input

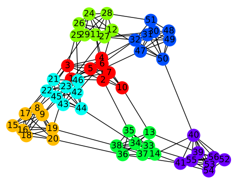

Figure 2: Division

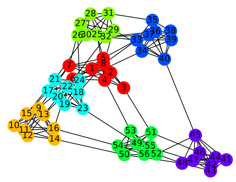

Figure 3: Relabelling

References

[1] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams.* Addison-Wesley Professional, 12th edition, 2009.

[2] Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. In *Proceedings of the 18th Annual European Conference on Algorithms: Part I*, ESA'10, page 549–560, Berlin, Heidelberg, 2010. Springer-Verlag.

[3] Shin-ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *Proceedings of the 30th International Design Automation Conference*, DAC '93, page 272–277, New York, NY, USA, 1993. Association for Computing Machinery.

[4] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[5] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, Feb 2004.