IEMS-452 Matching Project Report

Zuyue Fu

June 8, 2020

Abstract

We solve matching problem in this project using combinatorial optimization method. Our algorithm is out-of-core, which ensures that at any point of the algorithm, we do not process more than 30% of the edges in a graph. Our code is available at https://github.com/wuyuup/iems-452-proj.git.

1 Introduction

Max cardinality matching has a long history, and many existing algorithms solve the problem exactly, e.g., blossom algorithm. Due to the increasing sizes of graphs considered in recent tasks, algorithms that are out-of-core are required. In this course project, we design such an out-of-core algorithm, which will access at most 30% of edges at any point in the implementation of our algorithm.

2 Methodology

Our method is a heuristic-based augmenting path algorithm. In specific, our method consists of two parts. First, we implement a greedy algorithm , where we keep adding edges to the matching set until no further edges can be added. Second, repeatedly, we randomly load 30% edges of the graph to find augmenting paths for the current matching set. As the number of repetition grows, such a random choice of 30% edges of the graph ensures that we can finally find all available augmenting paths with a high probability. We summarize the idea in Algorithm 1.

Algorithm 1 Heuristic-based Augmenting Path Algorithm

1: **Input:** Graph $G = (V, E)$, number of iteration `niter`
2: **Output:** Matching M
3: $M \leftarrow$ `greedy`(G)
4: **for** $n = 1, 2, \ldots,$ `niter` **do**
5: Randomly select edge set E' from E such that $|E'| \leq 0.3|E|$, let $G' \leftarrow (V, E')$
6: $P \leftarrow$ `augmenting_path`(G', M)
7: $M \leftarrow M \Delta P$
8: **end for**

In Line 3 of Algorithm 1, we use the greedy algorithm `greedy` with G as the input. Such a greedy algorithm `greedy` is easily formalized given previous description; thus, we omit its details here. Similarly, in Line 6 of Algorithm 1, we use the augmenting path algorithm `augmenting_path` with partial graph G' and current matching M as inputs.

It is worthwhile to remark that the implementation of Line 5 can be in an out-of-core fashion. In specific, one may randomly read 30% edges in the graph from the disk, and only save these 30% edges in the memory.

3 Numerical Experiment

We use graph data from https://northwestern.box.com/s/pz1at6ilfl323sxnkulbmsh6kde4z3fw, which contains 21 graphs with sizes of optimal matching vary from 0 to 10272. Among these graphs, the numbers of rows and columns of the adjacency matrix in lp_e226 do not matching. To deal with this graph, we fill the adjacency matrix with zeros to match the dimension.

To obtain the accuracy (i.e., optimality gap) of Algorithm 1, we use the package NetworkX (Hagberg et al., 2008) to obtain the size of the optimal matching, which is reported in Table 1. As required, NetworkX is only used to obtain the optimal matching, and is not used for any other purpose.

We report the accuracy and standard deviations of Algorithm 1 with different choices of `niter`.

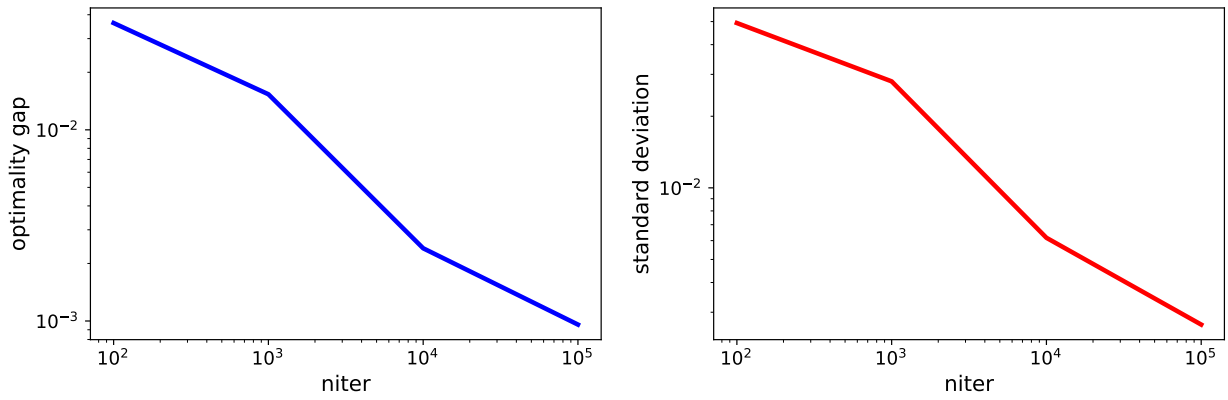

Figure 1: Optimality gaps and standard deviations of Algorithm 1 with `niter` $\in \{10^2, 10^3, 10^4, 10^5\}$.

From Figure 1, the choice `niter` $= 10^5$ achieves the best gap 0.096% with standard deviation 0.00266. For the completeness of this report, we report the accuracy with `niter` $= 10^5$ for each graph in Table 1.

4 Concluding Remarks

Such a method in Algorithm 1 does not guarantee that we will find the optimal matching even when `niter` goes to infinity, since we do not deal with possible blossoms in the graph. However, such a method does achieve great results on the given set of graphs, as we show in §3.

References

Hagberg, A., Swart, P. and S Chult, D. (2008). Exploring network structure, dynamics, and function using networkx. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

graph	opt.	ours	acc.
sphere3	129	129	0
poli	792	792	0
msc01440	720	720	0
mark3jac020sc	4554	4551	0.000659
lshp_406	203	203	0
lp_e226	200	200	0
dwt_72	32	32	0
dwt_2680	1340	1339	0.000746
dwt_198	99	99	0
can_62	29	29	0
bcsstm26	0	0	0
bcsstm02	0	0	0
bcsstm01	0	0	0
bcsstk05	76	76	0
bcspwr10	2576	2545	0.012
bcspwr01	17	17	0
bayer04	10272	10237	0.00341
b2_ss	544	544	0
G17	400	400	0
G15	400	400	0
662_bus	306	305	0.00327

Table 1: Accuracy for each graph with `niter` $= 10^5$.