A Simple Acceleration Method for the Louvain Algorithm

Naoto Ozaki, Hiroshi Tezuka, Mary Inaba*
Graduate School of Information Science and Technology, University of Tokyo, Tokyo, Japan.

* Corresponding author. Email: mary@ci.i.u-tokyo.ac.jp
Manuscript submitted October 10, 2015; accepted June 3, 2016.
doi: 10.17706/ijcee.2016.8.3.207-218

Abstract: The Louvain algorithm is well known for its high speed for detecting community structure in networks. In this paper, first, we analyze the Louvain algorithm as the preliminary experiment to uncover the processes that cause wasted computational time and their characteristics. Then based on this, we propose the Louvain Prune algorithm. The experiments show that the Louvain Prune algorithm significantly reduces computational time by up to 90%, and retains almost the same quality as the original Louvain algorithm.

Key words: Community detection, complex networks, Louvain algorithm, Heuristic, pruning.

1. Introduction

Networks are common in various domains such as biology, neurology, and especially social network analysis. It is known that those networks, called Complex networks, have common characteristics, degree distributions, high clustering, and a low average path length. Nodes often cluster together in dense groups, called community in the study of complex networks. Detecting these communities helps us understand their network characteristics such as the related functions of metabolites and groups of people who have a silar background and so on. Therefore, a lot of community detection algorithms have been proposed [1], and they often use modularity, one of the most popular objective functions in community detection algorithms [2]. The original modularity optimization algorithm makes a dendrogram then decides the cutting point at which modularity is maximized [2]. It is too slow for large graphs since it is running in $O(n^3)$, where n is the number of nodes and m is the number of edges. In fact, by optimizing modularity, we were able to obtain a good partition, but it is NP-complete in respect to maximum modularity [3]. Therefore, a lot of approximation modularity based algorithms have been proposed [1], [4], [5], and the one of them, the Louvain algorithm [6] that is believed to be running in $O(m)$ is well known as the one of the fastest and most effective algorithm. After this algorithm was proposed, some improved methods for the Louvain algorithm were proposed [7]-[9]. Additionally, other objective functions were also proposed [10], [11] since it is known that modularity has a problem called resolution limit in which modularity could be unable to find certain small communities [12].

The main contribution of this paper is a new easy-to-implement simple acceleration algorithm for the Louvain algorithm, the Louvain Prune algorithm. This enables the detection of communities faster while maintaining quality in synthetic networks and large real networks. We first describe the Louvain algorithm and the inefficient process observed in preliminary experiments using large real networks. Then, based on this, we propose the Louvain Prune algorithm. Finally, we show the results in which our proposal algorithm,

the Louvain Prune algorithm, significantly reduces computational time by up to about 90% and retains almost the same quality on networks including synthetic networks and real networks.

2. Related Work

2.1. Modularity

Modularity [2] is a widely used objective function to measure the strength of the division of a network into communities, and is normalized between -1 to 1. Modularity is defined as

$$Q(C)=\frac{1}{2m}\sum_{i\in V}\sum_{j\in V}(A_{ij}-\frac{k_i k_j}{2m})\delta(C_i,C_j) \tag{1}$$

where A_{ij} represents an element of the adjacent matrix, k_i is the total degree of node i, C_i is a label of the community to which node i is assigned and a $\delta(C_i,C_j)$ is the Kronecker delta symbol. Modularity tries to measure how many more internal edges are in the communities than in the expected one, with a random network preserving the degrees of the nodes.

2.2. Louvain Algorithm

The Louvain algorithm is a greedy modularity maximization algorithm, and is well known as the one of the fastest and most efficient community detection algorithm [6]. The input is a graph $G=(V,E)$ where V and E are the sets of nodes and edges. Community detection is performed by dividing graph G into clusters $C=\{V_1,V_2,\ \dots\ ,V_x\}$ and each V_i, a set of nodes, is called community. In this paper, we consider only non-overlapping nodes, so $V_i\cap V_j=\phi$ for all $i\neq j$ and all nodes have to be in a specific community.

Algorithm 1 shows the pseudo code of the Louvain algorithm. It is an iterative algorithm repeating until there is no additional improvement in modularity (line 6). Each iterations consists of two phases. It starts by initializing each node with its own community (line 3). In Phase 1, for each node in a graph, it computes the modularity gain ΔQ for all neighboring communities if the node moves (line 16-29). ΔQ indicates the modularity gain and is defined as

$$\Delta Q=\left[\frac{\Sigma_{in}+k_{i,in}}{2m}-\left(\frac{\Sigma_{tot}+k_i}{2m}\right)^2\right]-\left[\frac{\Sigma_{in}}{2m}-\left(\frac{\Sigma_{tot}}{2m}\right)^2-\left(\frac{k_i}{2m}\right)^2\right] \tag{2}$$

where Σ_{in} is the sum of the weights of the links inside the community to which the node i is assigned, Σ_{tot} is the sum of the weights of the links incident to nodes in the community, and $k_{i,in}$ is the sum of the weights of the links from i to nodes in the community which is same with the community of node i. However, the following simplified expression (3) is used in the practical implementation of the Louvain algorithm written in C++ by E. Lefebvre, one of the authors of [6].

$$\Delta Q=e_{iC}-\frac{k_i\Sigma_{tot}}{2m} \tag{3}$$

The term e_{iC} is the sum of the weights of the links between node i and community C. The node belongs to the neighboring community that gives the maximum gain in modularity. Phase 1 is repeated until a local maximum modularity is reached. In the Phase 2, all communities are collapsed to the vertices to create a new graph (line 33). Internal community edges are collapsed into a single self-looping edge, and the weight is the sum of the edge weights of all internal community edges in the community. Multiple edges between every two communities are collapsed into a single edge, and the weight is the sum of the edges between them.

2.3. Random Neighbor Louvain Algorithm

V. A. Traag suggested the Random Neighbor Louvain Algorithm (hereafter RNL) as the faster version of the Louvain algorithm [6]. Although the idea is so simple, it speeds up the Louvain algorithm roughly 2-3 times on synthetic networks resembling real networks and retains almost same quality if the graph has clear community structure. In MoveNodes, Phase 1, function, the RNL randomly choose the community of the neighbor node instead of considering all neighboring communities. Even though it is likely to lower the quality, it makes almost no effect to the quality since many neighbors are likely to belong to the same community in the initial partition. It means that even if we randomly choose the neighbor node, it is in same community with high probability. This idea significantly reduces computational time because even though the Louvain algorithm takes $O(k)$ to find the neighboring node that gives maximum modularity gain, the RNL takes a constant time $O(1)$ even if the nodes has high degree.

```
1:  function LouvainAlgorithm(Graph G)
2:    G' = G
3:    C the index of community of each nodes of G'
4:    Initialize each nodes with its own community
5:    q = −∞
6:    while q < Q(G',C) do
7:      q = Q(G',C)
8:      C = MoveNodes(G')     //Phase 1
9:      G' = Aggregate(G', C)  //Phase 2
10:     C = put each node of G'
                    in its own community
11:   end while
12:   return G'
13: end function

14: function MoveNodes(Graph G)
15:   C the index of communities
                    for each nodes of G
16:   while one or more nodes are moved do
17:     for random v ∈ V(G) do
18:       best_q = −∞
19:       best_c = community of v
20:       for all neighboring nodes n of v do
21:         gain_q = ΔQ between v and n
22:         if best_q < gain_q < then
23:           best_q = gain_q
24:           best_c = community of n
25:         end if
26:       end for
27:       C = Place v in the best_q
28:     end for
29:   end while
30:   return C
31: end function

32: function Aggregate(Graph G, Partition C)
33:   G' = aggregate nodes which are in
                    same community based on
      C
34:   return G'
35: end function
```

Algorithm 1. Louvain algorithm.

3. Observation of the Louvain Algorithm

In this section we first clarify the inefficient process of the Louvain algorithm and characteristics of the process through the input graph. Second, we consider the nodes that have the potential to change their community in the future.

3.1. The Inefficient Process of the Louvain Algorithm

Even though it is already known that the Louvain algorithm spends a huge amount of time on Phase 1 of the first iteration [6], [13], we must find the more specific process in order to optimize the algorithm.

The Louvain algorithm repeats phase 1 until there are no modularity improvements. Fig. 1 shows a case that represents the modularity transition and the computation time per Phase 1 loop on the large real network. It shows us that, although the modularity is dramatically increasing for some of the loops at the beginning, ultimately, the improvements per loop. However, the algorithm takes a relatively long time even for low quality improvements, and it is clearly inefficient in terms of computational time. This is caused by computing ΔQ for all neighboring nodes of all nodes of the graph per Phase 1 loop although only a very few nodes change their community after several Phase 1 loops, and it is assumed that a large number of Phase 1 loops makes the algorithm slower. Fig. 2 shows the high correlation between the total number of Phase 1 loops (hereafter P1L) and the computational time per loop on synthetic graphs generated by the LFR benchmark, widely used to evaluate community detection algorithms [14].

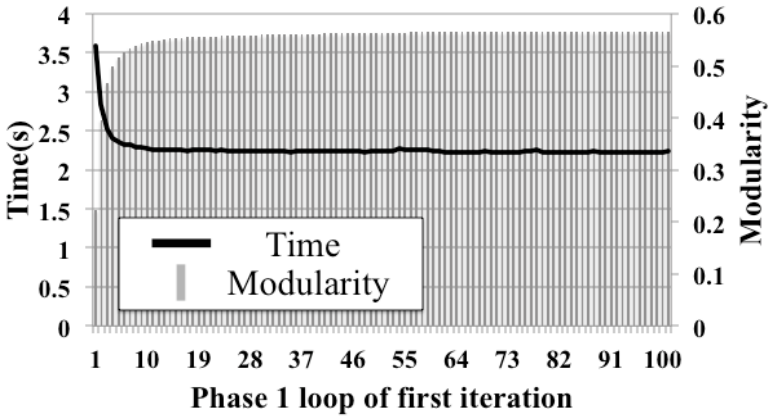


Fig. 1. Computational time and modularity per Phase 1 loop.

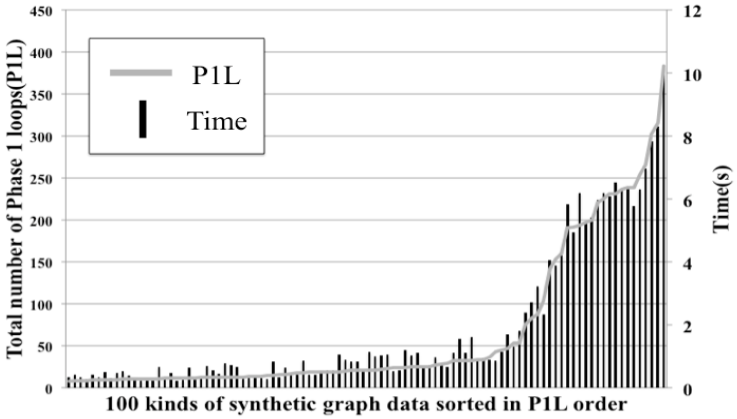


Fig. 2. Correlation between P1L and computational time for the Louvain algorithm.

The parameters of the synthetic graphs are set as same value except mixing parameter that is, the ratio between the external degree of a node and the total degree of the node. This indicates that the computational time of the Louvain algorithm highly depends on P1L and P1L apparently depends on the characteristics of the input graph.

Additionally, we clarified the some characteristics that cause an increase in P1L by running the algorithm on many kinds of synthetic graphs generated by LFR benchmark. The benchmark enables us to generate networks and adjust the parameters of the graph such as the graph size, average and maximum degree distribution, minimum and maximum community size, community overlap, and the mixing parameter. We generated 4096 graphs that included all possible combinations of the parameters shown in Table 1.

Table 1. Parameters of the Generated Graphs

Property	Parameter
Graph size	10000, 20000, 40000, 80000
Average degree	10, 20, 40, 80
Maximum degree	200, 400, 800, 1600
Minimum community size	10, 100, 300, 500
Maximum community size	1000, 1500, 3000, 5000
Mixing parameter	0.2, 0.4, 0.6, 0.8
Others	Undirected and Unweighted and No overlap

Then we ran the Louvain algorithm for them and counted the P1L on the first iteration. We used the C++ implementation program provided by E. Lefebvre, one of the authors of the paper of the Louvain algorithm for the experiments. All of the results are shown in Fig. 3-Fig. 5, 100% stacked column charts. The horizontal axis corresponds to the 4096 synthetic graphs sorted in P1L and it is divided into 16 parts. Considering the results in Fig. 3-Fig. 5, the worst graph for the algorithm has characteristics such as (A) communities mixed with each other, (B) high average degree size, and (C) low maximum degree size. Interestingly, Fig. 3-Fig. 5 indicates that the average degree parameter doesn't linearly correlate with P1L although the large average degree distribution creates a large P1L eventually. Therefore, in order to accelerate while retaining accuracy, it is necessary to find the nodes that have the potential for changing community in the next Phase 1 loop and compute the process for only them in Phase 1. Parameters of the synthetic graphs are set as same value except mixing parameter that is, the ratio between the external degree of a node and the total degree of the node. This indicates that the computational time of the Louvain algorithm highly depends on P1L and P1L apparently depends on the characteristics of the input graph.

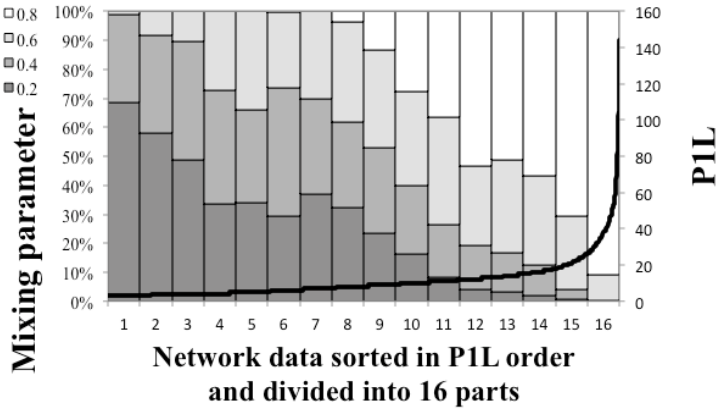


Fig. 3. Correlation between mixing parameter and P1L.

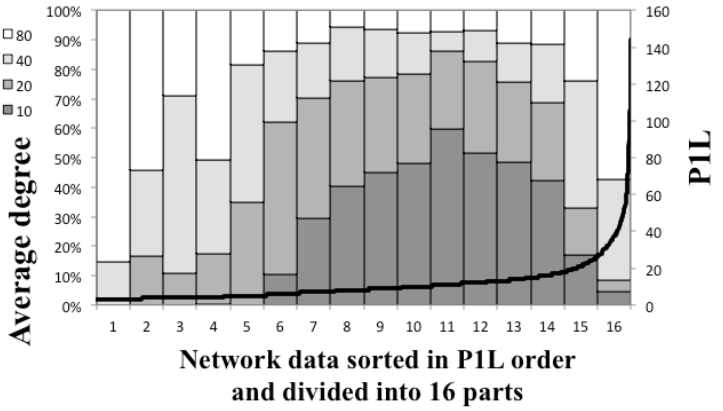


Fig. 4. Correlation between average degree and P1L.

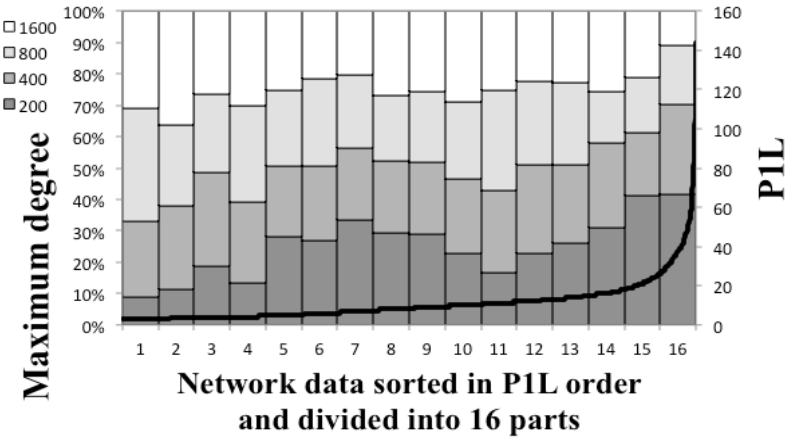


Fig. 5. Correlation between maximum degree and P1L.

3.2. Nodes That Have Potential to Change Community

As we described in the previous subsection, the process for computing ΔQ for all neighboring nodes of all nodes of the graph per Phase 1 loop takes too much time even for low quality improvements. If we can identify the nodes that will change their community in the next Phase 1 loop and consider only them in Phase 1, the calculation time will be significantly reduced. Although it is impossible to identify them accurately, we can narrow down the nodes that have the potential to change their community in the future by considering expression (3). Each values of ΔQ is changed by the community transition of other nodes. However, one node's community transition effects ΔQ for only the nodes of its neighbor, the nodes of its neighboring communities and the nodes of its new community. Therefore, we can mark the nodes that have higher ΔQ into other communities than previously when nodes change community. Therefore, considering only these in Phase 1 should reduces the inefficiencies within the process. In order to clarify the nodes that have the potential to change community in the future, we consider the one of the cases shown in Fig. 6 in which the node i changed the community from X to Y. There are four groups of nodes having this potential.

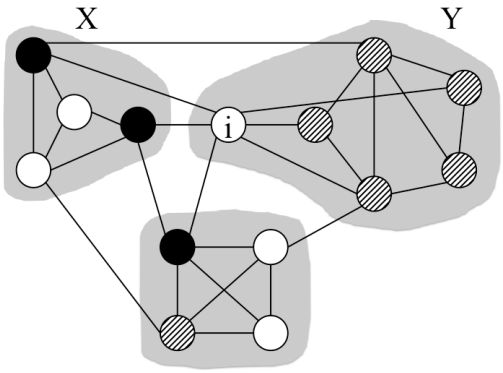


Fig. 6. Node i changed the community from X to Y.

(1) Neighboring nodes of the node i that aren't in communities Y. Their ΔQ into nodes in community X increases because of e_{iC}. They are shown as black nodes in Fig. 6.

(2) Neighboring nodes of the node i that are in community Y. They decrease ΔQ into their current community Y because the new node of community Y, the node i, increases Σ_{tot}.

(3) Neighboring nodes of community X that don't have links to the node i. Their ΔQ into community X

increases because of the decrease in Σ_{tot}.

(4) The nodes in community Y that don't have links to the node i. Their ΔQ into community Y decreases because Σ_{tot} increases.

In Phase 1, only the nodes in these four groups could affect the modularity. Therefore, the computational time will be reduced while maintaining the quality by considering only the nodes in these four groups.

4. Proposed Louvain Prune Algorithm

In this section, we propose the Louvain Prune algorithm which reduces computational time by up to 90% compared with the original algorithm, and it retains quality. As we described in the observation section, the Louvain algorithm spends too much time improving the slight quality in Phase 1 because the algorithm computes ΔQ for all neighboring nodes of all nodes of the graph per Phase 1 loop although only very few nodes change their community after several Phase 1 loops. The computational time will be reduced if we compute ΔQ for all neighboring nodes of only the nodes of the four groups described in the observation section. However, this produces not small overhead to find the all nodes of the four groups. Therefore, for the acceleration, we consider only the nodes of the Group 1 because they are the most influential for ΔQ among the groups. Surprisingly, this limitation almost doesn't affect the quality.

```
1:  function LouvainPruneMoveNodes(Graph G)
2:    C the index of communities for each nodes of G
3:    P = V(G)
4:    while P ≠ ∅ do
5:      v = random node x ∈ P
6:      P = P − {v}
7:      best_q = −∞
8:      best_c = community of v
9:      for all neighboring nodes n of v do
10:       gain_q = ΔQ between v and n
11:       if best_q < gain_q then
12:         best_q = gain_q
13:         best_c = community of n
14:       end if
15:     end for
16:     C = Place v the best_c
17:     for all neighboring nodes n of v do
18:       if n is not in community of v then
19:         P = P + {n}
20:       end if
21:     end for
22:   end while
23:   return C
24: end function
```

Algorithm 2. Louvain Prune algorithm.

Our proposed Louvain Prune algorithm replaces the MoveNodes function in Algorithm 1 with the function described in Algorithm 2. It computes ΔQ for only nodes in P which have nodes that have a high

potential to change their community in the future. Initially, in first Phase 1 loop, all nodes have the same potential to change their community, and P is initialized with all nodes of the graph (line 3). Then a random node v is selected and removed from P (line 5-6). Line 7-16 is the same process as the original one to find the one neighboring community which offers maximum modularity gain to change its community. In line 17-21, only neighboring nodes of v that are in the same community as v are added to P. This selection significantly reduces the computational time of the algorithm. These processes repeat until P doesn't have any nodes, and this means no nodes have a high potential to change their community (line 4).

In the case shown in Fig. 6 in which the node i changed its community from X to Y, even though not only black nodes but also striped nodes have the potential to change their community, we added only the black nodes to P because they have a higher potential than the rest.

5. Experimental Results

5.1. Experimental Environment and Program

We compare the computational times and the modularity of the Louvain Prune algorithm with the original Louvain algorithm by running a series of experiments on a Ubuntu 14.04.2 with an intel Core i7-3630QM CPU 2.40GHz and 16GB RAM. We use the Louvain algorithm program provided by E. Lefebvre and the Louvain Prune algorithm program as a modification of the original program.

5.2. Data Sets

We use the synthetic graphs generated by the LFR benchmark in order to experiment with various kinds of networks and the real large graphs provided as the datasets in the Koblenz Network Collection [15], [16]. The 4096 synthetic graphs include all combinations of Table 1. The real graph properties are shown in Table 2.

Table 2. The Real Data Used in the Experiment

Network	Node	Edge	Property
Amazon(MDS)	334,863	925,872	The co-purchase network of Amazon based on the "customers who bought this also bought" feature.
California	1,965,206	2,766,607	The road network of the State of California in the United States of America.
Youtube	3,223,589	9,375,374	The social network of YouTube users and their friendship connections.
Skitter	1,696,415	11,095,298	The network of an autonomous systems on the Internet connected to each other from the Skitter project.
Flickr links	2,302,925	33,140,017	The social network of Flickr users and their connections.
US patents	3,774,768	165,189,470	The citation network of patents registered with the United States Patent and Trademark Office.
Orkut	11,514,053	327,037,487	The social network of Orkut users and their connections.

5.3. Experimental Results — Efficiency

Almost all of the results show that the Louvain Prune algorithm accelerates the algorithm, and we confirmed that the Louvain Prune algorithm reduces computational time by up to 90% compared with the Louvain algorithm. Fig. 7 shows the speed improvement of the Louvain Prune algorithm on synthetic networks. The result demonstrates that the Louvain Prune algorithm accelerates on almost all of the synthetic graphs we used and the larger P1L with greater acceleration since the Louvain Prune algorithm avoids the process considered unnecessary as we described in the observation section. Table 3 indicates that the Louvain Prune algorithm reduces the computational time on, not only synthetic graphs, but also on real graphs. In particular, we achieve a 10-fold speed increase in the US patents network. Surprisingly, for even a small P1L such as California, the Louvain Prune algorithm appears to be more effective on real graphs than

on synthetic networks in most cases and it indicates the existence of other influences of unknown network characteristics. Fig. 8 shows the calculation time of the Louvain algorithm and the Louvain Prune algorithm per phase 1 loops in the first iteration on the US patents network. The calculation time of the Louvain Prune algorithm is significantly decreased after the first loop because of pruning effect even though the Louvain algorithm takes 2 seconds or more per loop.

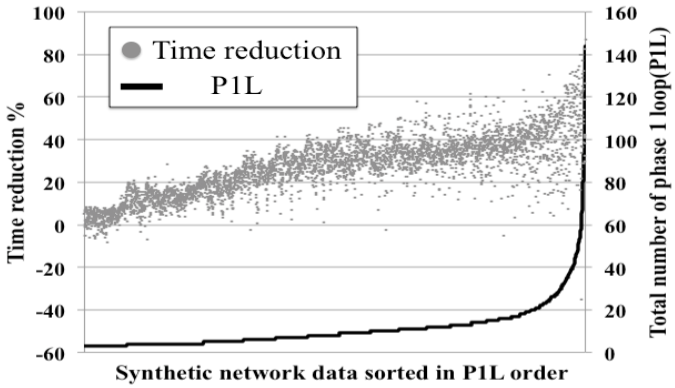


Fig. 7. LP results on 4096 synthetic networks.

Table 3. Numerical Results of the Real Graphs

	Flickr	Amazon	Youtube	Skitter	Orkut	California	US patents
Louvain algorithm	0.643174 12.9s	0.925653 1.9s	0.718395 7.2s	0.827941 14.5s	0.667979 249.9s	0.992300 16.5s	0.811286 241.1s
Louvain Prune algorithm	0.643158 4.4s	0.925906 0.6s	0.718101 1.9s	0.828270 3.4s	0.666621 56.5s	0.992255 3.0s	0.811141 22.9s
P1L(Louvain/ Louvain Prune)	17/17	15/15	18/16	21/21	60/60	13/8	101/111
Modularity Error	0.000024%	0.000273%	0.000409%	0.000397%	0.002032%	0.000045%	0.000178%
Time reduction	66%	69%	74%	77%	78%	82%	91%

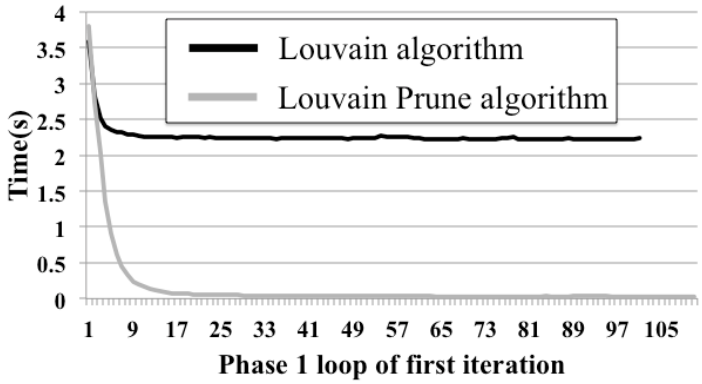


Fig. 8. Calculation time of the Louvain algorithm and the Louvain Prune algorithm per phase 1 loops of the first iteration on US patents network.

5.4. Experimental results — Accuracy

All of results show the Louvain Prune algorithm retains nearly the same modularity as the Louvain algorithm even though we did not consider all of the nodes that have the potential to change community in the future. This implies that considering only part of the nodes in the Phase 1 loop is sufficient to maintain quality. Fig. 9 shows the modularity difference between the Louvain algorithm and the Louvain Prune algorithm by calculating the Error percentage [13]. Error E is defined as

$$E = abs(\frac{q_{original} - q_{prune}}{q_{original}}) \quad (4)$$

where $q_{original}$ and q_{prune} represents the final modularity obtained from the Louvain algorithm and the Louvain Prune algorithm respectively. The result indicates that the larger P1L, the larger the error, but this is only less than 0.18%. Fig. 10 shows that, not only final modularity, but also the transition of the modularity appear to be almost the same with the original one.

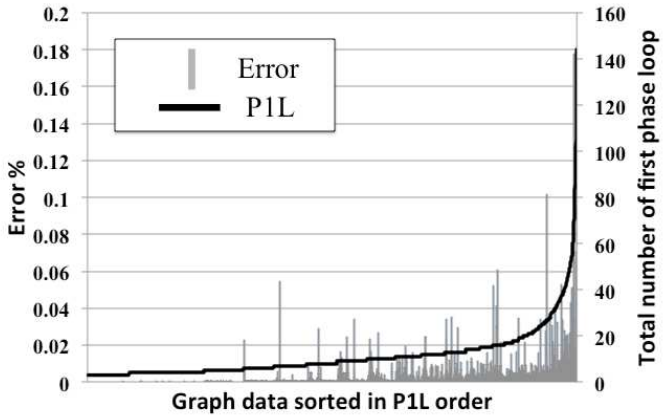


Fig. 9. Modularity error percentage of the Louvain Prune algorithm compared with the Louvain Prune algorithm on the synthetic graphs.

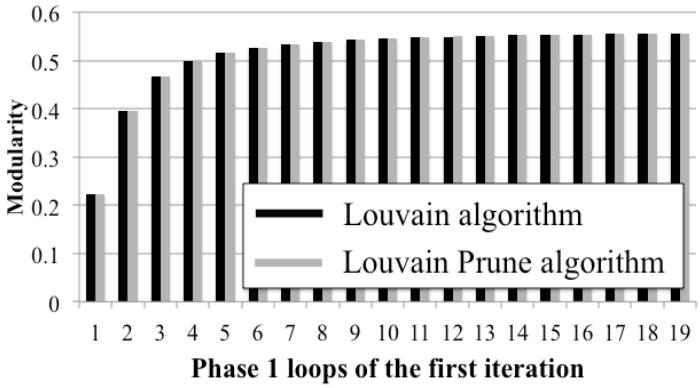


Fig. 10. Modularity transition between loop 1 and 19 per phase 1 loop in first iteration of the Louvain algorithm and the Louvain Prune algorithm on US patents network.

6. Concluding Remarks

The Louvain algorithm is well-known as the one of the fastest and most efficient community detection algorithms. In this paper, we clarified the network characteristics which result in more computational time in the Louvain algorithm, and we proposed a simple acceleration algorithm, the Louvain Prune algorithm, based on the Louvain algorithm. The Louvain Prune algorithm enables us to analyze large networks up to 10 times faster by pruning the inefficient process and the experimental results appears to show that the Louvain Prune algorithm can accelerate without quality deterioration on almost every kinds of network generated by the LFR benchmark and on all real networks we tested.

We are planning further evaluation of the Louvain Prune algorithm. First, we intended to clarify why the Louvain Prune algorithm can maintain almost the same quality even though it doesn't consider all of the

cases that have the potential to improve quality. Second, we will take into account other objective functions such as Surprise, Significance by replacing them with Modularity since it is currently known that Modularity has a problem called resolution limit. Third, we intend to the Louvain Prune algorithm with other acceleration techniques such as [7] and [8] to explore further improvements.

References

[1] Fortunato, S. (2010). Community detection in graphs. *Phys. Rep., 486,* 75-174.

[2] Newman, M. E. J., & Girvan, M. (2004). Finding and evaluating community structure in networks, *Phys. Rev. E., 69,* 026113.

[3] Brandes, U., Delling, D., Gaertler, M., Goerke, R., Hoefer, M., Nikoloski, Z., & Wagner, D. (2006). Maximizing modularity is hard. *arXiv: Physics, 0608255*. Retrieved October 10, 2015, from http://arxiv.org/abs/physics/0608255

[4] Waltman, L., & Van Eck, N. J. (2013). A smart local moving algorithm for large-scale modularity-based community detection. *Europian Physical Journal B, 86(11),* 471.

[5] Ovelgönne, M., & Geyer-Schulz, A. (2013). An ensemble learning strategy for graph clustering. *Proceedings of 10th DIMACS implementation Challenge Graph Partitioning and Graph Clustering: Vol. 588* (pp. 187-206).

[6] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment, 10008(10),* 6.

[7] Peng, C., Kolda, T. G., & Pinar, A. (2014). Accelerating community detection by using K-core subgraphs. *arXiv:1403.2226*. Retrieved October 10, 2015, from http://arxiv.org/abs/1403.2226

[8] Traag, V. A. (2015). Faster unfolding of communities: Speeding up the Louvain algorithm. *arXiv:1503.01322*. Retrieved October 10, 2015, from http://arxiv.org/abs/1403.2226

[9] Gach, O., Hao, J.-K. (2014). Improving the Louvain algorithm for community detection with modularity maximization. *arXiv:1406.2518*. Retrieved October 10, 2015, from http://arxiv.org/abs/1503.01322

[10] Traag, V. A., Krings, G., & Van Dooren, P. (2013). Significant scales in community structure. *Nature Scientific Reports: Vol. 3*(p. 2930).

[11] Aldecoa R., & Marín, I. (2013). Surprise maximization reveals the community structure of complex networks. *Nature Scientific Reports: Vol. 3* (p. 1060).

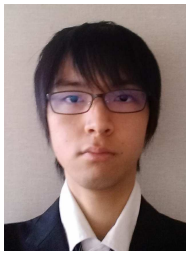
[12] Fortunato S., & Barthélemy, M. (2007). Resolution limit in community detection. *Natl. Acad. Sci. U. S. A., 104(1*), 36–41.

[13] Wickramaarachchi, C., Frincu, M., Small, P., & Prasanna, V. K. (2014). Fast parallel algorithm for unfolding of communities in large graphs. *Proceedings of High Perform. Extrem. Comput. Conf. (HPEC)* (pp. 1–6).

[14] Lancichinetti, A., Fortunato, S., & Radicchi, F. (2008). Benchmark graphs for testing community detection algorithms. *Phys. Rev., E 78 046110*.

[15] konect network dataset – KONECT. (2015). From http://konect.uni-koblenz.de/

[16] Kunegis, J. (2013). KONECT — the Koblenz network collection. *Proceedings of Int. Conf. on World Wide Web Companion*(pp. 1343-1350).



Naoto Ozaki is a master course student at the Graduate School of Information Science and Technology, University of Tokyo, Japan. His research interest is in community detection algorithms.

Hiroshi Tezuka is a project researcher at the Department of Creative Informatics Graduate School of Information Science and Technology, University of Tokyo, Japan. His research interests include operating systems and user interface.

Mary Inaba is an associate professor at the Department of Creative Informatics, Graduate School of Information Science and Technology, University of Tokyo, Japan. She was a research associate at the Faculty of Science, University of Tokyo during 1996-1999, and a lecturer during 1999-2002. She received the Ph.D. from the Department of Information Science, Faculty of Science, University of Tokyo in February 1999. Her research interests include computational geometry, networking, and high performance computing.