UDC 004.021:004.75

REQUIREMENTS TO METHODS OF GRAPH CLUSTERING AT THE AIM OF SOLVING THE SHORTEST PATH PROBLEM

O.N. Karasik

Tech Lead at ISsoft Solutions (part of Coherent Solutions) in Minsk, Belarus, PhD in Technical Science
karasik.oleg.nikolaevich@gmail.com

A.A. Prihozhy

Professor at the Computer and System Software Department,
Doctor of Technical Sciences,
Full Professor
Belarusian National Technical University
prihozhy@yahoo.com

O.N. Karasik

Tech Lead at ISsoft Solutions (part of Coherent Solutions) in Minsk, Belarus; PhD in Technical Science (2019). Interested in parallel computing on multi-core and multi-processor systems.

A.A. Prihozhy

Full professor at the Computer and system software department of Belarusian national technical university, doctor of science (1999) and full professor (2001). His research interests include programming and hardware description languages, parallelizing compilers, and computer aided design techniques and tools for software and hardware at logic, high and system levels, and for incompletely specified logical systems. He has over 300 publications in Eastern and Western Europe, USA and Canada. Such worldwide publishers as IEEE, Springer, Kluwer Academic Publishers, World Scientific and others have published his works.

Abstract. *In this paper we considered utilization of graph clustering results in scope of solving all-pairs shortest path problem by means of blocked all-pairs shortest paths algorithm with unequally sized blocks. We defined a set of requirements for the results of graph clustering based on the inner working of the blocked algorithm. We have done an analysis of two existing, well-known graph clustering algorithms (Walktrap and Spinglass) to verify if existing clustering algorithm can produce results consumable by blocked all-pairs shortest path algorithm with unequally sized blocks. Our experiments show, that both algorithms can be used to produce compatible results, however, in different contexts.*

Keywords: *All-pairs shortest path problem, blocked algorithms, graph clustering, unequally sized blocks.*

Introduction. The problem of finding the shortest paths between vertices in a directed, weighted graphs, has numerous applications in many real-life domains, like traffics, computer networks, social networks, computer games, hardware benchmarking and so on [1–3]. The shortest paths problem can be formulate differently depending of what shortest paths have to be found: a problem of finding a shortest path between a pair of vertices (**S**ingle **P**air **S**hortest **P**ath), a problem of finding all shortest paths between one vertex and the rest of the vertices (**S**ingle **S**ource **S**hortest **P**ath or **S**ingle **S**ink **S**hortest **P**ath) and a problem of finding all pairs of shortest paths between all pairs of vertices in the graph (**A**ll **P**airs **S**hortest **P**ath). Computational complexity of the shortest paths problem depends on multiple parameters such as size of the graph (number of vertices), density of the graph (number of edges), edge weight (real or integer, positive or negative) and problem formulation. For each of the formulation and combination of parameters there are classical algorithms [4–7] developed.

However, effective solution of the problem on large graphs with classical algorithms might take a significant portion of time because they weren't designed to run on modern, multi-core, multi-processors, or heterogeneous systems. That is why, to reduce execution time modern algorithms (or modernized versions of the classic algorithms) try to make use of hardware parallelism on single chip, powerful accelerators, distributed systems and pre-computed ahead of time information about or combination of those [8–10].

In our previous works we concentrated on improving solution of all-pairs shortest path problem using blocked version of Floyd-Warshall algorithm [11] by exploiting parallelism, CPU caching, data dependencies and novel algorithms to recalculate blocks [12–15]. In this paper we are focused on defining requirements to use pre-computed information about target graph's clusters (clustering) to speed up execution.

Algorithms. In this section we provide a concise description of Floyd-Warshall [7] and blocked Floyd-Warshall [11] algorithms to lay the foundation for further analysis.

Floyd-Warshall algorithm operates on a cost adjacency matrix $D[N \times N]$, where N is the number of vertices in a graph, and element $D_{i,j}$ contains a weight of the edge between vertices i and j. The algorithm recalculates matrix D in N iterations, where every iteration consists of picking the vertex k (essentially all vertex of matrix D are picked) and checking if any of the paths between vertices i and j can be shortened through vertex k (see Figure 1).

```
int N = ...
...
function algorithm(matrix D)
  for k = 0 to N do
    for i = 0 to N do
      for j = 0 to N do
        D[i,j] = min(D[i,j], D[i,k] + D[k,j])
      end
    end
  end
end function
```

Figure 1. Pseudocode of original Floyd-Warshall algorithm

Blocked Floyd-Warshall algorithm extends Floyd-Warshall algorithm by splitting the matrix D into $S \times S$ blocks, effectively creating a matrix $B[M{\times}M]$, where $M * S = N$. The algorithm recalculates matrix B in M iterations, where every iteration consists of three phases: calculating the a single «diagonal» block, which has dependency on itself, $2 \cdot (M-1)$ «cross» blocks, which have dependency on themselves and the "diagonal" block and $(M-1)^2$ «peripheral» blocks, which have dependency on the corresponding vertical and horizontal «cross» blocks (see *Figure 2*).

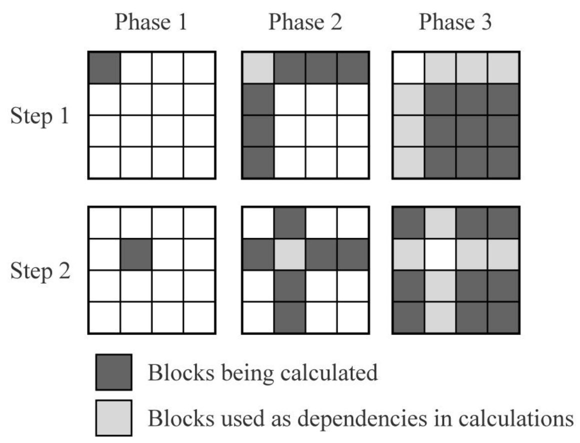


Figure 2. Illustration of calculation phases of Blocked Floyd-Warshall algorithm on first two iterations (steps)

All the blocks are calculated using the same procedure which always accepts three blocks (see *Figure 3*).

```
int M = ...
int S = ...
...
function blocked_algorithm(block_matrix B)
  for m = 0 to M do
    proc(B[m,m], B[m,m], B[m,m]);
    for i = 0     to m - 1 do
      proc(B[i,m], B[i,m], B[m,m]); proc(B[m,i], B[m,m], B[m,i]);
    end
    for i = m + 1 to M - 1 do
      proc(B[i,m], B[i,m], B[m,m]); proc(B[m,i], B[m,m], B[m,i]);
    end
    for i = 0     to m - 1 do
      for j = 0     to m - 1 do proc(B[i,j], B[i,m], B[m,j]); end
      for j = m + 1 to M - 1 do proc(B[i,j], B[i,m], B[m,j]); end
    end
    for i = m + 1 to M - 1 do
      for j = 0     to m - 1 do proc(B[i,j], B[i,m], B[m,j]); end
      for j = m + 1 to M - 1 do proc(B[i,j], B[i,m], B[m,j]); end
    end
  end
end function
function proc(B1, B2, B3)
  for k = 0 to S do
    for i = 0 to S do
      for j = 0 to S do
        B1[i,j] = min(B1[i,j], B2[i,k] + B3[k,j])
      end
    end
  end
end function
```

Figure 3. Pseudocodes of blocked Floyd-Warshall algorithm (blocked_algorithm) and block calculation procedure (proc).

The blocked version of the algorithms takes advantage of the CPU caching by improving data localization during block recalculations i.e. no more than three blocks participate in calculation, which makes it possible to fit all of them into CPU cache and reduce number of main memory accesses.

Clustering. In this section we provide a concise description of the graph clustering to lay a foundation for further analysis.

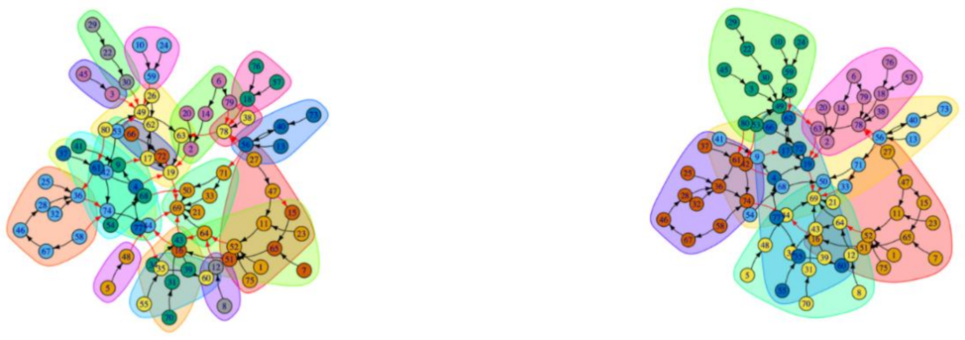
Graph clustering is an operation of grouping vertices of the graph into clusters (or communities) by taking into consideration the structure of the graph (edge adjacency, distance or

similarity between vertices, type of the graph and so on) in such a way that there should be many edges within each cluster and few between the clusters [16]. In terms of directed graphs, which are the main audience of the shortest path problem, graph clustering provides information about highly inter-connected and sparsely-connected sub-graphs, bridge vertices (i.e. vertices which constitute edges between clusters) and bridge edges (i.e. edges between clusters).
Depending on the graph, clustering algorithm and its parameters (or in cases when clustering algorithm uses any kind of randomization, even a particular run) the clustering results can have significant difference in number of clusters, number of vertices in clusters and number of bridge vertices and edges. In Table 1 you can find clustering results of the small (80 vertex) randomly generated, directed, connected graph using Walktrap [17] and Spinglass [18] clustering algorithms and on the Figure 4 you can see a visualization of execution of Walktrap, with *walks* set to 7, and Spinglass, with *spins* set to 7, algorithms.

Table 1. Results of the clustering of a randomly generated, directed, connected graph of 80 vertices and 94 edges using Walktrap and Spinglass clustering algorithms with different sets of parameters.

Walktrap				Spinglass			
Walks	Clusters	Bridge vertices	Bridge edges	Spins	Clusters	Bridge vertices	Bridge edges
2	17	24	26	2	2	7	7
3	19	26	28	3	3	10	10
4	19	27	29	4	4	11	11
5	19	26	29	5	5	13	13
6	18	27	28	6	6	15	15
7	20	29	30	7	7	17	17

Table 1 demonstrates how results of graph clustering differ in number of clusters and bridge vertices / edges between different algorithms and their parameters.



a) b)

Figure 4. Visualization of graph clustering of a randomly generated, directed, connected graph of 80 vertices using Walktrap, with walks set to 7, (a) and Spinglass, with spins set to 7, (b) algorithms. Red arrows represent bridge edges, filled shapes and vertex colors represent clusters.

Analysis. In this section we perform an analysis of the results of execution of Walktrap and Spinglass graph clustering algorithms to define requirements to methods of graph clustering and input graphs based on how these results can be used to improve execution of blocked all-pairs shortest paths algorithm.

In appliance to blocked all-pairs shortest path algorithms the information from graph clustering can potentially be used to improve algorithm's performance. However, because of how blocks are recalculated (see *Figure 2*) these optimizations (which might rely on advanced knowledge of the graph's layout) are possible if clusters are matched to blocks i.e. every cluster is represented by a "diagonal" block. In classic variation of the blocked all-pairs shortest paths algorithm (see *Figure 3*), all the blocks must be of same size. This is a very strict requirement, which can hardly be matched by any of graph clustering algorithms. Luckily, this requirement can be relaxed as demonstrated in [19] by using blocks of unequal sizes. With an ability to use blocks of unequal sizes. Matching of clusters to "diagonal" blocks requires a rearrangement of rows and columns of cost adjacency matrix (before splitting it to blocks) to align all vertices of clusters around matrix diagonal. However, for the matching be successful and do not introduce a degradation instead of optimization, clusters must satisfy the following requirements:

1 *Clusters must not be too «small» or too «large»*[4]. Having a significant number of «small» clusters will result in a significant number of small blocks and CPU cache underutilization. Having large block results in contrary will reduce effect of the CPU cache and will result in a significant main memory pressure (as demonstrated in [20]).

2 *Clusters must not be too interconnected.* Having too many bridge vertices or bridge edges renders the makes the whole purpose of using clusters useless.

To understand, if the above requirements can be fulfilled, by existing graph clustering algorithm, we performed a set of experiments using two, well-known algorithms (Walktrap and Spinglass) on three randomly generated, connected graphs with artificially generated clusters (RCGWC). Information about all graphs is presented in Table 2. To verify how algorithms treat artificially generated clusters, during the generation of the graphs, we have recorded the ranges of vertices constituting to all clusters (see Table 3).

Table 2. Configuration of the experimental graphs, where RCG are randomly generated, connected graphs of different density and RCGWC are randomly generated connected graphs with artificially generated clusters.

Graph	Vertices	Edges	Bridge Vertices	Bridge Edges	Clusters
RCGWC-1	1200	49192	223	298	12
RCGWC-2	1200	46923	408	572	10
RCGWC-3	1200	56567	743	1415	10

We have executed all algorithms on all the experimental graphs with parameters varying from 2 to 12 walks and 2 to 12 spins for Walktrap and Spinglass algorithms respectively.

During the experiments, Walktrap algorithm has precisely identified all artificially generated clusters in all experimental graphs independently of number of walks. This demonstrates that results of graph clustering using Walktrap algorithm are very sensitive to graph's layout and can be utilized when the layout matches the requirements. The executing time of the Walktrap algorithm varied from 0.3 to 1.5 seconds[5] (depending on the number of walks – more walks more time).

[4] The «small» and «large» here are relative to the characteristics of the executing CPU cache memory, where small is less than ¼ of Level 1 cache and "large" is more than ½ of Level 2 cache.

[5] All clustering algorithms were executed on M1 Ultra Mac Studio using R language (igraph library).

Table 3. Configuration of artificially generated clusters of all RCGWC graphs.

Cluster	RCGWC-1		RCGWC-2		RCGWC-3	
	Vertices	Vertices Range	Vertices	Vertices Range	Vertices	Vertices Range
1	77	0 – 76	122	0 – 121	116	0 – 115
2	170	77 – 246	154	122 – 275	134	116 – 249
3	84	247 – 330	111	276 – 386	58	250 – 307
4	55	331 – 385	166	387 – 552	61	308 – 368
5	64	386 – 449	146	553 – 698	167	369 – 535
6	143	450 – 592	161	699 – 859	56	536 – 591
7	53	593 – 645	115	860 – 974	155	592 – 746
8	143	646 – 788	49	975 – 1023	141	747 – 887
9	95	789 – 883	96	1024 – 1119	162	888 – 1049
10	109	884 – 992	80	1120 – 1199	150	1050 – 1199
11	120	993 – 1112				
12	87	1113 – 1199				

During experiments (see Table 4), Spinglass algorithm produced various results. In most cases, the number of clusters was equal to number of spins (which is expected because of how algorithm works). However, when algorithm was executed against RCGWC-2 and RCGWC-3 graphs, which have 10 artificial clusters each, with number of spins set to 11 and 12 it still identified 10 clusters (not 11 or 12). Produces clusters only partially matched the structure of artificially generated clusters but despise this they weren't highly interconnected or of significantly different sizes (see Table 5):

1 The number of bridge vertices varied:
   - from 210 to 299 (where predefined was 223) for RCGWC-1
   - from 270 to 425 (where predefined was 408) for RCGWC-2
   - from 544 to 750 (where predefined was 743) for RCGWC-3

2 The number of bridge edges varied:
   - from 327 to 496 (where predefined was 298) for RCGWC-1
   - from 323 to 711 (where predefined was 572) for RCGWC-2
   - from 889 to 1522 (where predefined was 1415) for RCGWC-3

In combination, results presented in Table 4 and Table 5 experimentally demonstrate that results of graph clustering using Spinglass algorithm can be utilized with other parameters to optimally split the cost-adjacency matrix into block even when the natural layout of the graph doesn't matches the requirements. The executing time of the Spinglass algorithm varied from 9 to 11 seconds (depending on the number of spins – more spins more time).

Table 4. Experimental results of the Spinglass algorithm executed on all RCGWC graphs depicting number of clusters, number of bridge vertices and edges in relation to spins parameter.

Spins	RCGWC-1			RCGWC-2			RCGWC-3		
	Clusters	Bridge Vertices	Bridge Edges	Clusters	Bridge Vertices	Bridge Edges	Clusters	Bridge Vertices	Bridge Edges
2	2	224	353	2	270	323	2	544	889
3	3	210	327	3	360	506	3	618	1074
4	4	229	381	4	376	575	4	691	1288
5	5	253	448	5	392	584	5	697	1324

End of table 4

Spins	RCGWC-1			RCGWC-2			RCGWC-3		
	Clusters	Bridge Vertices	Bridge Edges	Clusters	Bridge Vertices	Bridge Edges	Clusters	Bridge Vertices	Bridge Edges
6	6	222	328	6	391	617	6	714	1349
7	7	299	497	7	405	622	7	724	1361
8	8	273	475	8	398	619	8	739	1433
9	9	268	473	9	414	711	8	747	1522
10	10	250	360	10	417	662	10	750	1492
11	11	272	421	10	416	634	10	747	1514
12	12	274	452	10	425	673	10	749	1516

Table 5. Experimental results of Spinglass algorithm on RCGWC-1 graph depicting number of vertices in clusters in relation to spins parameter value.

Spins/Cluster	2	3	4	5	6	7	8	9	10	11	12
1	592	380	344	171	309	145	144	144	143	121	52
2	608	361	325	197	143	144	162	143	136	83	143
3		459	359	258	170	178	170	138	87	143	77
4			172	364	143	170	144	170	78	143	95
5				210	250	196	191	110	120	170	170
6					185	247	173	121	118	87	120
7						120	96	96	109	95	65
8							120	130	95	77	86
9								148	144	65	110
10									170	107	143
11										109	86
12											53

Conclusion. We have defined a set of requirements for methods of graph clustering and produced clustering results to be useful for blocked all-pairs shortest paths algorithms. We performed an experimental analysis of two, well-known graph clustering algorithms (Walktrap and Spinglass) to verify if produced results match the requirements and can be used in blocked all-pairs shortest paths with unequal block sizes, or this task requires a specialized clustering algorithm which must be found or developed.

The experiments demonstrated that Walktrap algorithm can be used when natural layout of the graph matches the requirements. Spinglass algorithm has demonstrated an ability to produce variable results depending on the parameters. This shows that algorithm can be used even when natural layout of the graph doesn't match the requirements. Through these experiments we have confirmed that existing methods of graph clustering can produce results compatible with defined requirements, which makes implementation of the potential optimizations based on the availability of clustering information a prospective direction for future research.

References

[1] Comparison of HPC Architectures for Computing All-Pairs Shortest Paths. Intel Xeon Phi KNL vs NVIDIA Pascal / M. Costanzo [et al.]. – Springer, 2020. – P. 37–49.

[2] Anu, P. Finding All-Pairs Shortest Path for a Large-Scale Transportation Network Using Parallel Floyd-Warshall and Parallel Dijkstra Algorithms / P. Anu, M. G. (Kumar) // Journal of Computing in Civil Engineering. – 2013. – Vol. 27, №. 3. – P. 263–273.

[3] Schrijver, A. On the history of the shortest path problem / A. Schrijver // Documenta Mathematica. – 2012. – Vol. 17, №. 1. – P. 155–167.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs / E. W. Dijkstra // Numerische Mathematik. – 1959. – Vol. 1, №. 1. – P. 269–271.

[5] Bellman, R. On a routing problem / R. Bellman // Quarterly of applied mathematics. – 1958. – Vol. 16, №. 1. – P. 87–90.

[6] Johnson, D. B. Efficient algorithms for shortest paths in sparse networks / D. B. Johnson // Journal of the ACM (JACM). – 1977. – Vol. 24, №. 1. – P. 1–13.

[7] Floyd, R. W. Algorithm 97: Shortest Path / R. W. Floyd // Communications of the ACM. – 1962. – Vol. 5, №. 6. – P. 345-.

[8] Efficient multi-GPU computation of all-pairs shortest paths / H. Djidjev [et al.]. – IEEE, 2014. – P. 360–369.

[9] The all-pair shortest-path problem in shared-memory heterogeneous systems / H. Ortega-Arranz [et al.] // High-Performance Computing on Complex Environments. – 2013. – P. 283–299.

[10] A scalable parallelization of all-pairs shortest path algorithm for a high performance cluster environment / T. Srinivasan [et al.]. – IEEE, 2007. – P. 1–8.

[11] Venkataraman, G. A Blocked All-Pairs Shortest Paths Algorithm / G. Venkataraman, S. Sahni, S. Mukhopadhyaya // Journal of Experimental Algorithmics (JEA). – 2003. – Vol. 8. – P. 857–874.

[12] Karasik, O. Tuning block-parallel all-pairs shortest path algorithm for efficient multi-core implementation / O. Karasik, A. Prihozhy // System analysis and applied information science. – 2022. – №. 3. – P. 57–65.

[13] Karasik, O. Profiling of energy consumption by algorithms of shortest paths search in large dense graphs / O. Karasik, A. Prihozhy // Big Data and Advanced Analytics: сб. материалов IX Междунар. науч.-практ. конф., Минск. – Минск: BSUIR, 2023. – P. 44–50.

[14] Prihozhy, A. Influence of shortest path algorithms on energy consumption of multi-core processors / A. Prihozhy, K. Oleg // System analysis and applied information science. – 2023. – №. 2. – P. 4–12.

[15] Карасик, О. Н. Кооперативный многопоточный планировщик и блочно-параллельные алгоритмы решения задач на многоядерных системах / О. Н. Карасик. – Белорусский государственный университет информатики и радиоэлектроники, 2019.

[16] Schaeffer, S. E. Graph clustering / S. E. Schaeffer // Computer science review. – 2007. – Vol. 1, №. 1. – P. 27–64.

[17] Pons, P. Computing communities in large networks using random walks / P. Pons, M. Latapy. – Springer, 2005. – P. 284–293.

[18] Reichardt, J. Statistical mechanics of community detection / J. Reichardt, S. Bornholdt // Physical review E. – 2006. – Vol. 74, №. 1. – P. 016110.

[19] Prihozhy, A. New blocked all-pairs shortest paths algorithms operating on blocks of unequal sizes / A. Prihozhy, O. Karasik // System analysis and applied information science. – 2023. – №. 4. – P. 4–13.

[20] Karasik, O. Parallel blocked all-pair shortest path algorithm: block size effect on cache operation in multi-core system / O. Karasik, A. Prihozhy // BIG DATA and Advanced Analytics: сб. материалов VIII Междунар. науч.-практ. конф., Минск, 11-12 мая 2022. – Минск: Бестпринт, 2022. – P. 28–38.

author's contribution

Authors to make an equivalent contribution.

ТРЕБОВАНИЯ К МЕТОДАМ КЛАСТЕРИЗАЦИИ ГРАФОВ С ЦЕЛЬЮ РЕШЕНИЯ ЗАДАЧИ О КРАТЧАЙШИХ ПУТЯХ

О.Н. Карасик
Ведущий инженер иностранного производственного унитарного предприятия «ИССОФТ СОЛЮШЕНЗ» (ПВТ, г. Минск), к.т.н.

А.А. Прихожий
Профессор кафедры «Программное обеспечение информационных систем и технологий» Белорусского национального технического университета, д.т.н., профессор

Аннотация. В данной статье рассматривается возможность использования результатов кластеризации графа для решения задачи поиска всех кратчайших путей в графе при помощи блочного алгоритма поиска кратчайших путей, использующего блоки неравного размера. В статье определяются требования к результатам кластеризации графа на основании принципа работы блочного алгоритма поиска кратчайших путей. Проводится исследование двух, широко-известных алгоритмов кластеризации графа – *Walktrap* и *Spinglass*, с целью выяснения возможности использования результатов их работы блочным алгоритмом. Экспериментальные исследования показывают, что выбранные алгоритмы способны произвести результаты совместимые с определенными в статье требованиями, однако совместимость этих результатов во многом зависим от исходного графа и заданных параметров алгоритмов.

Ключевые слова: Кластеризация графа, блочные алгоритмы, поиск кратчайших путей, блоки неравного размера.