UDC 004.021:004.75

BLOCKED ALGORITHM OF SHORTEST PATHS SEARCH IN SPARSE GRAPHS PARTITIONED INTO UNEQUALLY SIZED CLUSTERS

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
karasik.oleg.nikolaevich@gmail.com

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.

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.

Abstract. *In this paper we consider the problem of searching shortest paths between all pairs of vertices of a directed weighted sparse graph which is partitioned into clusters by finding dense weakly connected subgraphs. We address this problem by developing new block-based algorithms that describe the shortest paths by matrices of blocks of unequal sizes corresponding to the sizes of the graph clusters. These algorithms extend the capabilities of known existing algorithms using blocks of equal size (such as the blocked algorithms of Floyd-Warshall family) with respect to adequate graph modeling of real networks of different purposes, and with respect to efficient use of parallelism and computational resources of multiprocessor systems and multi-core processors. The blocked algorithm of finding shortest paths in sparse large size graphs partitioned into clusters that is proposed in this paper reduces, on the one hand, the amount of memory used, and, on the other hand, reduces the number of block recalculations. Diagonal blocks describe shortest paths within clusters, non-diagonal compact blocks describe non numerous weighted arcs connecting clusters. Shortest paths between vertices of different clusters are computed in real time. The memory consumption is reduced compared to the Floyd-Warshall algorithm to a number of times equal to the number of clusters. In order to reduce the number of block recalculations, a new operation is introduced to accurately compute the shortest path between vertices of one cluster, passing through the vertices and edges of another cluster, as well as through the edges connecting the clusters. Applying this operation alone allows us to find solutions that introduce a small error (a few percent) in the lengths of the shortest paths when the weights of edges between clusters are small, and allows us to find exact solutions when the weights of these edges are increased. Accurate solutions can be obtained for sparse graphs modeling road, computer, and other networks.*

Keywords: *sparse graph, cluster, APSP problem, blocked algorithm, unequally sized blocks, heterogeneous system.*

Introduction. The problem of finding shortest and longest paths for all pairs of vertices in a large sparse weighted graph [1 – 6] has many application domains. Recently the emergence of

heterogeneous parallel computing systems [7 – 9] has increased interest in this problem. Many competitive algorithms are developed for various types of graphs and for different formulations of the problem: between two vertices; between the source (sink) and each other vertex (single source and single sink – SSSP); between each pair of vertices (all pairs shortest paths– APSP); to meet the requirement that all graph vertices must be listed on the path, etc.

The paper considers the APSP problem and algorithms [10, 11] of solving it. Two families of the APSP algorithms exist: 1) based on the Dijkstra SSSP-algorithm [1]; 2) based on the Floyd-Warshall APSP-algorithm [2]. The first family includes the Dijkstra algorithm [1], the Bellman-Ford algorithm [12], the Johnson algorithm [13], the Harish and Narayanan algorithm [14], and others [15]. The second family includes among others the Floyd-Warshall (*FW*) algorithm [2], the blocked Floyd-Warshall algorithm (*BFW*) proposed in [7, 10, 11] by Katz, Venkataraman and others, the graph extension-based algorithm (*GEA*) and the heterogeneous blocked APSP algorithm (*HBAPSP*) both proposed by Prihozhy and Karasik in [16 – 18]. The algorithms can be parallelised by OpenMP [19]. The results as follows have been obtained based on the idea of using blocks: a recursive blocked *FW* algorithm [10]; efficient usage of GPUs [7 – 9]; solving sparse graph scaling problem [20]; optimization of data allocation in hierarchical memory [21]; improving cache performance for APSP [11, 17, 22]; a cooperative threaded algorithm [23, 24]; selection of the optimal block-size [25]; reducing energy consumption [26]; search for shortest paths using dataflow networks of actors [27, 28]. In work [5], a method of inferring new blocked algorithms which divide the input graph into unequal subgraphs and divide the matrix of shortest path distances into blocks of unequal sizes has been proposed.

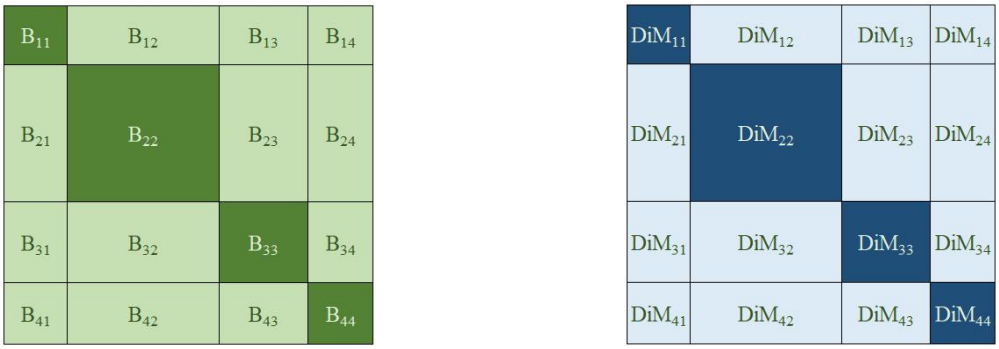
The key contribution of the paper is a fast and memory efficient blocked algorithm of computing the shortest paths within unequally sized clusters of a large sparse graph and computing the shortest paths between vertices of different clusters in real time.

Blocked all-pairs shortest paths algorithm for unequally sized blocks. Let $G = (V, E)$ be a simple directed graph with real edge-weights consisting of a set V, $|V| = N$ of vertices numbered 1 through N and a set E of edges. Let W be a cost adjacency matrix for G. So, $w(i, i) = 0$, $1 \le i \le N$; $w(i, j)$ is the cost (weight) of edge (i, j) if $(i, j) \in E$ and $w(i, j) = \infty$ if $i \ne j$ and $(i, j) \notin E$. Let d_{ij} be a length of shortest path from vertex i to vertex j, and D be a matrix of distances between all pairs of vertices $i, j \in V$, $i \ne j$. Let P be a matrix whose element p_{ij} is a vertex that is previous for vertex j in a path from i to j. The objective of an APSP-algorithm is to compute the D and P matrices for a given graph G.

In work [5], we have proposed to decompose the graph G into subgraphs (clusters) and decompose the matrix B into blocks of unequal sizes defined by vector $S = (S_1 \ldots S_M)$ (Figure 1,a). While M blocks are square on the principal diagonal of B (block B_{ii} has the $V_i \times V_i$ size), all other blocks are rectangular in general case (block B_{ij} has the $V_i \times V_j$ size for $i, j = 1 \ldots M$, $i \ne j$). All blocks in row i have the height of V_i, and all blocks in column j have the width of V_j. Matrix P of previous vertices in the shortest paths has the same structure.

At the aim of processing unequaly sized clusters, we extended the known blocked Floyd-Warshall algorithm *BFW* to the all-pairs shortest path algorithm *BFWUS* [5], which can handle a block-matrix B of unequally sized blocks. *BFWUS* is described by Algorithm 1. In a loop along m that performs M iterations, it recalculates each of M^2 blocks of matrix B, therefore, it carries out M^3 recalculations in total. In terms of vertex count, the time complexity of *BFWUS* is N^3 and the memory complexity is N^2 since each block has the layout (Figure 1,b) of matrix of shortest path distances (*DiM*).

Figure 2 illustrates the operation of *BFWUS* and depicts the order of calculating blocks.



a) b)

Figure 1. Blocked matrix *B* of shortest paths distances: a) diagonal blocks are square and non-diagonal blocks are rectangular; b) *BFWUS* represents all blocks by matrix of shortest path distances (*DiM*)

Algorithm 1: Blocked APSP algorithm accounting for blocks of unequal sizes (*BFWUS*)

Input: A number N of vertices in input graph
Input: A matrix $W[N\times N]$ of graph edge weights
Input: A vector $S = (S_1 \ldots S_M)$ of sizes of vertex subsets
Input: A number M of blocks per row (column)
Output: A blocked matrix $B[M\times M]$ of path distances
Output: A blocked matrix $P[M\times M]$ of previous vertices in shortest paths

```
for i, j ← 1 to N do
    if W(i, j) ≠ ∞ then P^init(i, j) ← i else P^init(i, j) ← undefined
B[M×M] ← W[N×N]    P[M×M] ← P^init[N×N]
for m ← 1 to M do
    USBC (S, B, P, m, m, m)                          // D0
    for v ← 1 to M do
        if v ≠ m then
            USBC (S, B, P, v, m, m)                  // C1
            USBC (S, B, P, m, m, v)                  // C2
    for v ← 1 to M do
        if v ≠ m then
            for u ← 1 to M do
                if u ≠ m then
                    USBC (S, B, P, v, m, u)          // P3
return B, P
```

Algorithm 2 describes a block-calculation procedure *USBC* which has a feature of processing blocks of unequal sizes. The algorithm inputs are metrices B and P which describe blocks of sizes defined by vector S. Indices v, m and u choose in matrix B three blocks $B_{v,u}$, $B_{v,m}$ and $B_{m,u}$ of which two or three can be identical. The indices choose similar blocks in matrix P. The sizes of blocks are $S_v \times S_u$, $S_v \times S_m$ and $S_m \times S_u$ respectively. *USBC* consists of three nested loops. It makes $S_m \cdot S_v \cdot S_u$ attempts to update the values of elements of blocks $B_{v,u}$ and $P_{v,u}$. The order of loops is essential. The loop along k must be the outer, it cannot be reordered with other loops.

The key difference *USBC* against Floyd-Warshall is that the non-diagonal blocks are rectangles but squares.

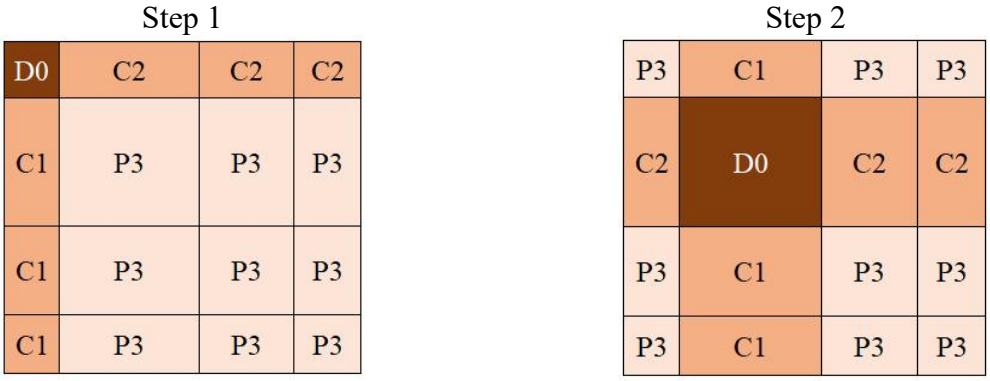


Figure 2. Illustration of *BFWUS* operation: cross moves from top-left to bottom-right corner of blocked matrix *B*; first, block *D*0 is calculated through itself; second, blocks *C*1 and *C*2 are calculated through *D*0; third, blocks *P*3 are calculated through *C*1 and *C*2

Algorithm 2: Calculation of unequally sized blocks (*USBC*)

Input: A vector S of sizes of graph vertex subsets
Input: A blocked matrix $B[M \times M]$ of path distances
Input: A blocked matrix $P[M \times M]$ of previous vertices in shortest paths
Input: Indices v, m and u of vertex subsets
Output: Recalculated block $B_{v,u}$ of matrix B
Output: Recalculated block $P_{v,u}$ of matrix P

```
for k ← 1 to S_m do
    for i ← 1 to S_v do
        for j ← 1 to S_u do
            sum ← B_{v,m}(i, k) + B_{m,u}(k, j)
            if B_{v,u}(i, j) > sum then
                B_{v,u}(i, j) ← sum
                P_{v,u}(i, j) ← P_{m,u}(k, j)
return B, P
```

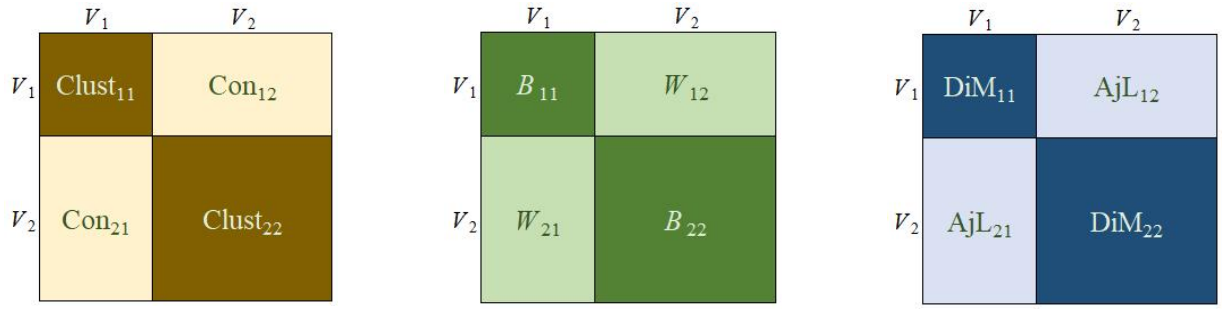
Computation of shortest paths between vertices within two clusters through neighbor cluster and interconnecting edges. In case of matrix B[2×2], the BFWUS algorithm calculates two diagonal and two non-diagonal blocks in the following way:

$$B_{11}[S_1 \times S_1] \leftarrow B_{11}[S_1 \times S_1] \otimes B_{11}[S_1 \times S_1] \quad (1)$$
$$B_{21}[S_2 \times S_1] \leftarrow B_{21}[S_2 \times S_1] \otimes B_{11}[S_1 \times S_1] \quad (2)$$
$$B_{12}[S_1 \times S_2] \leftarrow B_{11}[S_1 \times S_1] \otimes B_{12}[S_1 \times S_2] \quad (3)$$
$$B_{22}[S_2 \times S_2] \leftarrow B_{21}[S_2 \times S_1] \otimes B_{12}[S_1 \times S_2] \quad (4)$$
$$B_{22}[S_2 \times S_2] \leftarrow B_{22}[S_2 \times S_2] \otimes B_{22}[S_2 \times S_2] \quad (5)$$
$$B_{12}[S_1 \times S_2] \leftarrow B_{12}[S_1 \times S_2] \otimes B_{22}[S_2 \times S_2] \quad (6)$$
$$B_{21}[S_2 \times S_1] \leftarrow B_{22}[S_2 \times S_2] \otimes B_{21}[S_2 \times S_1] \quad (7)$$
$$B_{11}[S_1 \times S_1] \leftarrow B_{12}[S_1 \times S_2] \otimes B_{21}[S_2 \times S_1]. \quad (8)$$

Equations (1) – (4) calculate block B_{22} through block B_{11}. First, diagonal block B_{11} is calculated through itself using (1). Operator $\otimes$ denotes a matrix MIN-PLUS multiplication operation. Then blocks B_{21} and B_{12} are calculated through B_{11} using (2) and (3). After that, diagonal block B_{22} is calculated through blocks B_{21} and B_{12} using (4). Equations (5) – (8) perform similar operations in opposite direction, i.e. from block B_{22} to block B_{11}. It is important that three operations are needed to calculate B_{22} through B_{11}, i.e. (2), (3) and (4). Similarly, three operations are needed to calculate B_{11} through B_{22}, i.e. (6), (7) and (8). To accomplish it, two intermediate blocks are additionally calculated, i.e. B_{21} and B_{12}. For large sparse graphs, *BFWUS* requires a huge amount of memory space and processor time. For very large graphs it is unjustified and practically unacceptable.

We propose a new method of computing B_{22} through B_{11} (and similarly B_{11} through B_{22}). The method allows to account for features of sparse graphs with clustered vertices, to reduce the amount of consumed memory space, to decrease the number of matrix MIN-PLUS operations executed over blocks.

Let two clusters $Clust_{11}$ and $Clust_{22}$ divide the vertex set V of graph G into two subsets V_1 and V_2 of unequal sizes S_1 and S_2 (Figure 3,a). The vertices of clusters $Clust_{11}$ and $Clust_{22}$ are connected by edges from Con_{12} and Con_{21}. As a result, we obtain a matrix B consisting of four blocks (Figure 3,b). Blocks B_{11} and B_{22} initially describe the internal weighted edges of the clusters, and then describe the shortest path lengths between vertices of set V_1 and between vertices of set V_2. Sparse blocks W_{12} and W_{21} describe weighted edges connecting vertices of V_1 to vertices of V_2 and vice versa respectively. Figure 3,c shows that blocks B_{11} and B_{22} are placed in memory as matrices of distances (*DiM*) using row-major memory layout, and blocks W_{12} and W_{21} are placed as adjacent lists (*AjL*).



a) b) c)

Figure 3. Dividing graph to clusters: a) diagonal blocks are clusters and non-diagonal blocks describe connections between clusters; b) blocks B_{11} and B_{22} describe shortest path lengths and blocks W_{12} and W_{21} describe weighted edges; c) diagonal blocks are represented by matrix of distances (*DiM*) and non-diagonal blocks are represented by adjacent lists (*AjL*)

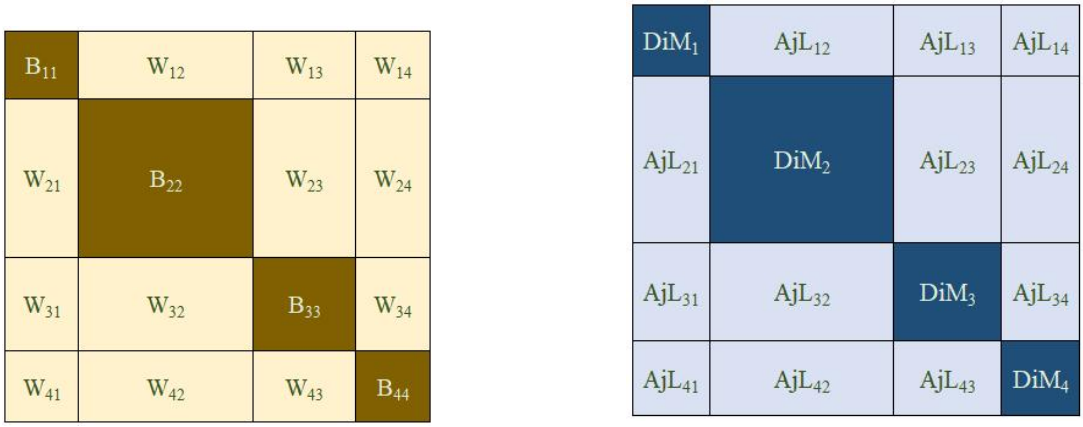
Our method of computing block B_{22} through block B_{11} and vice versa consists in performing the following five operations:

$$B_{11} \leftarrow Diagonal(B_{11}) \quad (9)$$
$$B_{22} \leftarrow BlockThroughBlock(B_{22}, W_{21}, B_{11}, W_{12}) \quad (10)$$
$$B_{22} \leftarrow Diagonal(B_{22}) \quad (11)$$
$$B_{11} \leftarrow BlockThroughBlock(B_{11}, W_{12}, B_{22}, W_{21}) \quad (12)$$
$$B_{11} \leftarrow Diagonal\ (B_{11}) \quad (13)$$

Operation *Diagonal* (B_{ii}) calculates the shortest paths between all vertices of block B_{ii}; the paths can traverse through edges within B_{ii} and outside it. The operation can be performed using (1) or can preferably be implemented by the fast *GEA* algorithm proposed in [16, 17].

New operation *BlockThroughBlock*(B_{jj}, W_{ji}, B_{ii}, W_{ij}) calculates the shortest paths between vertices of block B_{jj}. It traverses through edges of block W_{ji}, then through vertices and edges of block B_{ii} and finally through edges of block W_{ij}. Since the edges of blocks W_{ji} and W_{ij} are not numerous in the sparse graph, the *BlockThroughBlock* operation is fast. When W_{ji} or W_{ij} is empty, *BlockThroughBlock* is not executed at all. Moreover, the shortest paths of these blocks do not need to be stored in memory, only edge descriptions are needed to store. This is a big advantage of our method which yields accurate solutions. The shortest paths between vertices of different clusters are calculated in real time. It is easier to compute the shortest paths between clusters after computing shortest paths within clusters.

Approximate fast algorithm of computing all-pairs shortest paths in clusters of sparse graph. We have developed an approximate APSP algorithm operating on graph clusters (AAPSPC) aiming at reduction of the consumed memory space and CPU time. Figure 4,a shows the content of matrix B and Figure 4,b depicts the matrix representation and placement in memory. Clusters are placed in the matrix principal diagonal, and interconnections of clusters are represented by graph edges (blocks of matrix W) allocated outside of principal diagonal. Clusters are represented by matrices of shortest path distances (DiM). Blocks of matrix W are represented as adjacent lists (AjL).



a) b)

Figure 4. Matrix *B* of shortest paths distances in clusters: a) diagonal blocks B_{ii} are clusters and non-diagonal blocks W_{ij} represent edges connecting clusters; b) each cluster is represented by distances matrix (*DiM*) and each non-diagonal block is represented by adjacent list (*AjL*)

Algorithm 4 specifies *AAPSPC*. Its input data are the vertex subsets corresponding to clusters, and their sizes. Matrices $B = W$ and P describing initial states of shortest paths in clusters are also input data. The algorithm output data are the completely calculated diagonal blocks of matrices B and P.

The algorithm consists of two loop nests of depth two. The first nest traverses the diagonal blocks from the left-top to right bottom corner of the matrices and calculates the shortest paths of each succeeding block through previous blocks. The second nest does the same work in reverse order. Totally M diagonal blocks are calculated. The overall number of the diagonal block recalculations *AAPSPC* performs is

$$Rec(B) = M \cdot (M + 1). \tag{14}$$

It is almost M times smaller than the number M^3 of block recalculations *BFWUS* performs.

Function *Diagonal**(S, B, P, d) is implemented using operation *Diagonal*(B_{dd}). Function *BlockThroughBlock**(S, B, P, d, p) is implemented using operation *BlockThroughBlock*(B_{dd}, W_{dp}, B_{pp}, W_{pd}).

Algorithm 4: Computing all-pairs shortest paths in clusters of sparse graphs (*AAPSPC*)

Input: Subsets $V = (V_i, \dots V_M)$ of set V of graph clustered vertices
Input: A vector $S = (S_i, \dots S_M)$ of sizes of graph vertex subsets
Input: A blocked matrix $B[M \times M]$ representing graph
Input: A blocked matrix $P[M \times M]$ of previous vertices in shortest paths
Output: Recalculated matrix $B[M \times M]$
Output: Recalculated matrix $P[M \times M]$

```
for d ← 1 to M do
    Diagonal'(S, B, P, d)
    if d > 1 then
        for p← 1 to d – 1 do
            BlockThroughBlock*(S, B, P, d, p)
for d ← M down to 1 do
    Diagonal'(S, B, P, d)
    if d > 1 then
        for p← d – 1 down to 1 do
            BlockThroughBlock*(S, B, P, p, d)
return B, P
```

The calculation of shortest paths between vertices of different clusters is carried out in real time through the earlier computing of shortest paths between vertices within clusters.

Let's estimate the volume of storage needed for allocation of matrix B in algorithms *BFWUS* and *AAPSPC*. *BFWUS* uses the *DiM* representation of all blocks, Therefore, $storage(BFWUS) = N^2$. The amount of storage consumed by *AAPSPC* can be estimated as

$$storage(AAPSPC) = storage(Clusters) + storage(Edges), \quad (15)$$

$$storage(Clusters) = \sum_{c=1}^{M} S_c^2 \quad (16)$$

where c is a cluster; M is the number of clusters; S_c is the number of vertices in cluster c; $storage(Edges)$ is the size of all non-diagonal blocks Wij describing interconnect edges. For sparse graphs, $storage(Edges)$ is not huge.

Let's consider the bounding case when all clusters have the same size $S_c = N / M$. Then the amount of storage they need for placement is

$$storage(Clusters) = \left(\frac{N}{M}\right)^2 \cdot M = {}^{N^2}/_{M} \quad (17)$$

It is M times smaller than *BFWUS* needs. If the clusters are of unequal sizes, the gain is reduced. We conclude that *AAPSPC* consumes a much smaller amount of memory compared to *BFWUS* in case of sparse graphs. It is a very big advantage in the case of searching for the shortest paths in very large sparse graphs.

At the same time, *AAPSPC* has a drawback: it considers only shortest paths that go from one cluster to another (maybe iteratively) and return to the former cluster. It accounts for not all paths passing through several clusters before returning to the source . It is important to know what inaccuracies arise in calculating the shortest path distances due to these restrictions. We have done experiments with *AAPSPC* with respect to the inaccuracies.

For instance, we provide results obtained on a directed weighted graph consisting of 4 clusters, 128 vertices and 1502 edges with the average weight of 55.54. Figure 5 shows that the inaccuracies in computing the shortest path lengths are small and vary in the range 1.1778 % down to 0.0 % depending on the weights of edges that connect clusters. The inaccuracies disappear at the average weight of 23 which is 41.4 % compared to the average weight of edge within clusters. *AAPSPC* yields an accurate shortest path distances between all pairs of vertices in all clusters for all interconnect edge weights larger than 41.4 %.

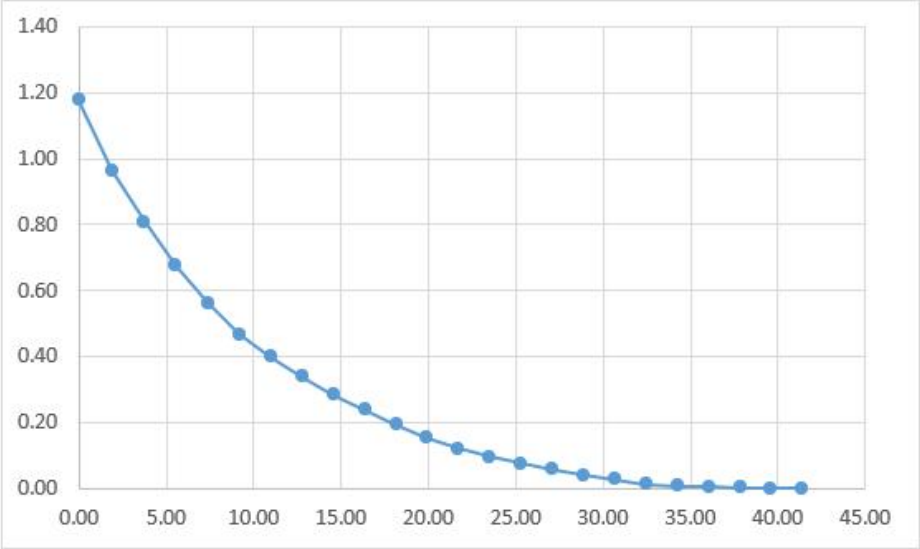


Figure 5. Inaccuracy (vertical axis, %) in computation of shortest path distances vs. share (horizontal axis, %) of average weight of edge between clusters in average weight of edge within clusters

Right reordering of clusters in matrix B can decrease the required number of block recalculations or / and reduce the inaccuracies in computations of shortest paths. Incorporating Dijkstra algorithm [1] or any algorithm of Dijkstra's family in *AAPSPC* is the way of avoiding the inaccuracies in shortest paths computation and obtaining accurate solutions.

Conclusion. The paper solves the all-pairs shortest paths problem on sparse graphs partitioned into clusters by finding dense weakly connected subgraphs. The approach we propose is based on recently published blocked algorithms which divide the graph into unequally sized subgraphs. We have developed an operation of computing the shortest paths between vertices of one cluster passing through vertices and edges of neighbor cluster, and through edges connecting the clusters. This allowed us to develop a fast and memory efficient approximate algorithm that first computes the shortest paths between vertices within each cluster and then computes shortest paths between vertices of different clusters in real time. The algorithm gives accurate solutions for road, computer and other networks in which the weights of edges connecting clusters are typically at least as large as the weights of edges within clusters.

References

[1] Dijkstra E. W. A note on two problems in connexion with graphs. Numerische Mathematik, 1959, vol. 1, no. 1, pp. 269–271.

[2] Floyd R.W. Algorithm 97: Shortest path. *Communications of the ACM*, 1962, no. 5 (6), p. 345.

[3] Glabowski M., Musznicki B., Nowak P. and Zwierzykowski P. Review and Performance Analysis of Shortest Path Problem Solving Algorithms. International Journal on Advances in Software, 2014, vol. 7, no. 1&2, pp. 20 – 30.

[4] Madkour A., Aref W. G., Rehman F. U., Rahman M. A., Basalamah S. A Survey of Shortest-Path Algorithms. ArXiv: 1705.02044v1 [cs.DS], 4 May 2017, 26 p.

[5] Prihozhy, A., Karasik, O. New blocked all-pairs shortest paths algorithms operating on blocks of unequal sizes. *System analysis and applied information science*. – 2023. – №. 4. – P. 4–13.
[6] Rahman A.-H. Ab, Prihozhy A., Mattavelli M. Pipeline synthesis and optimization of FPGA-based video
processing applications with CAL. EURASIP Journal on Image and Video Processing, vol. 2011:19, pp. 1–28.
[7] Katz G. J., Kider J. T. All-pairs shortest-paths for large graphs on the GPU. *GH'08: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware.* ACM, 2008, pp. 47-55.
[8] Ortega-Arranz H., Torres Y., Llanos D. R, and Escribano A. G. The all-pair shortest-path problem in shared-memory heterogeneous systems. High-Performance Computing on Complex Environments, 2013, pp. 283–299.
[9] Djidjev H., Thulasidasan S., Chapuis G., Andonov R. and Lavenier D. Efficient multi-GPU computation of all-pairs shortest paths. IEEE 28th International Parallel and Distributed Processing Symposium. IEEE, 2014, pp. 360–369.
[10] Venkataraman G., Sahni S., Mukhopadhyaya S. A Blocked All-Pairs Shortest Paths Algorithm. *Journal of Experimental Algorithmics* (JEA), 2003, vol 8, pp. 857 – 874.
[11] Park J. S., Penner M., and Prasanna V. K. Optimizing graph algorithms for improved cache performance. *IEEE Trans. on Parallel and Distributed Systems*, 2004, no. 15 (9), pp.769 – 782.
[12] Bellman R. E. On a routing problem. Quarterly of Applied Mathematics, 1958, vol. 16, no. 1, pp. 87–90.
[13] Johnson D. B. Efficient Algorithms for Shortest Paths in Sparse Networks. J. ACM, 1977, vol. 24 no. 1, pp. 1 – 13.
[14] Harish P., Narayanan P. J. Accelerating large graph algorithms on the GPU using CUDA. International conference on high-performance computing. Springer, 2007, pp. 197–208.
[15] Meyer U. and Sanders P. Δ-stepping: a parallelizable shortest path algorithm. Journal of Algorithms, vol. 49, no. 1, 2003, pp. 114–152.
[16] Prihozhy A. A., Karasik O. N. Heterogeneous blocked all-pairs shortest paths algorithm. *System analysis and applied information science*, 2017, no. 3, pp. 68 – 75. (In Russian).
[17] Prihozhy A.A., Karasik O.N. Inference of shortest path algorithms with spatial and temporal locality for big data processing. [*Big Data and Advanced Analytics*: proceedings of VIII international conference]. Minsk, Bestprint Publ., 2022, pp. 56 – 66.
[18] Prihozhy A. A., Karasik O. N. Advanced heterogeneous block-parallel all-pairs shortest path algorithm. *Proceedings of BSTU, issue 3, Physics and Mathematics. Informatics*, 2023, no. 1 (266), pp. 77–83. DOI: 10.52065/2520-6141-2023-266-1-13.
[19] Albalawi E., Thulasiraman P., Thulasiram R. Task Level Parallelization of All Pair Shortest Path Algorithm in OpenMP 3.0. *2nd International Conference on Advances in Computer Science and Engineering (CSE 2013)*. Los Angeles, CA, July 1 – 2, 2013, pp. 109 – 112.
[20] Yang S., Liu X., Wang Y., He X., Tan G. Fast All-Pairs Shortest Paths Algorithm in Large Sparse Graph. *ICS '23: Proceedings of the 37th International Conference on Supercomputing*, 2023, pp. 277–288.
[21] Prihozhy A. A. Simulation of direct mapped, k-way and fully associative cache on all-pairs shortest paths algorithms. *System analysis and applied information science*, 2019, no. 4, pp. 10 – 18.
[22] Prihozhy A. A. Optimization of data allocation in hierarchical memory for blocked shortest paths algorithms. *System analysis and applied information science*, 2021, no. 3, pp. 40 – 50.
[23] Prihozhy A.A., Karasik O.N. Cooperative block-parallel algorithms for task execution on multi-core system. *System analysis and applied information science*, 2015, no. 2, pp. 10–18.
[24] Karasik O. N., Prihozhy A. A. Threaded block-parallel algorithm for finding the shortest paths on graph. *Doklady BGUIR*, 2018, no. 2, pp. 77 – 84. (In Russian).
[25] Karasik O. N., Prihozhy A. A. Tuning block-parallel all-pairs shortest path algorithm for efficient multi-core implementation. *System analysis and applied information science*, 2022, no. 3, pp. 57 – 65.
[26] Prihozhy A.A., Karasik O.N. Influence of shortest path algorithms on energy consumption of multi-core processors. *System analysis and applied information science*, 2023, no. 2, pp. 4-12.
[27] Prihozhy A. A. Generation of shortest path search dataflow networks of actors for parallel multicore implementation. *Informatics*, 2023, vol. 20, no. 2, pp. 65−84.
[28] Прихожий, А.А. Кооперативная модель оптимизации выполнения потоков на многоядерной системе / А.А. Прихожий, О.Н. Карасик // Системный анализ и прикладная информатика. – 2014. – № 4. – С. 13-20

author's contribution

Authors to make an equivalent contribution.

БЛОЧНЫЙ АЛГОРИТМ ПОИСКА КРАТЧАЙШИХ ПУТЕЙ В РАЗРЕЖЕННЫХ ГРАФАХ, РАЗБИТЫХ НА КЛАСТЕРЫ НЕРАВНОГО РАЗМЕРА

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

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

Аннотация. В данной статье рассматривается задача поиска кратчайших путей между всеми парами вершин ориентированного взвешенного разреженного графа, который разбит на кластеры путем нахождения плотных слабосвязных подграфов. Мы решаем эту задачу, разрабатывая новые блочные алгоритмы, которые описывают кратчайшие пути матрицами блоков неравных размеров, соответствующих размерам кластеров графа. Эти алгоритмы расширяют возможности известных существующих алгоритмов, использующих блоки одинакового размера (таких как блочный алгоритм Флойда-Уоршелла) в части адекватного моделирования графов реальных сетей различного назначения, а также в части эффективного использования параллелизма и вычислительных ресурсов многопроцессорных систем и многоядерных процессоров. Предлагаемый в данной статье блочный алгоритм поиска кратчайших путей в разреженных графах большого размера, разбитых на кластеры, позволяет, с одной стороны, сократить объем используемой памяти, а с другой – уменьшить количество пересчетов блоков. Диагональные блоки описывают кратчайшие пути внутри кластеров, недиагональные блоки описывают немногочисленные взвешенные дуги, соединяющие кластеры. Кратчайшие пути между вершинами разных кластеров вычисляются в реальном времени. Потребление памяти по сравнению с алгоритмами семейства Флойда-Уоршелла уменьшается в число раз, равное количеству кластеров. Чтобы уменьшить количество пересчетов блоков, нами предложена новая операция, позволяющая точно вычислить кратчайшие пути между вершинами одного кластера, проходящие через вершины и ребра другого кластера, а также через ребра, соединяющие кластеры. Применение только этой операции позволяет построить алгоритмы, которые находят решения, вносящие небольшую ошибку (несколько процентов) в длины кратчайших путей при малых весах ребер между кластерами, и позволяет находить точные решения при увеличении весов этих ребер. Точные решения могут быть получены для разреженных графов, моделирующих дорожные, компьютерные и другие сети.

Ключевые слова: разреженный граф, кластер, кратчайшие пути, блочный алгоритм, блоки неравного размера, разнородная система.