УДК 004.4–004.9

A. A. Prihozhy[1], O. N. Karasik[2]
[1]Belarus National Technical University
[2]ISsoft Solutions (part of Coherent Solutions)

FAST SEARCH FOR SHORTEST PATHS IN LARGE SPARSE GRAPHS DIVIDED INTO CONNECTED DENSE CLUSTERS

The aim of the paper is to solve the problem of finding shortest paths between all pairs of vertices in simple directed weighted large sparse graphs. It is assumed that the graphs with positive and negative real weights of edges are decomposed into unequally sized weakly connected clusters. Since the problem has numerous practical applications in various domains, and the graphs may be too large, our goal is to speed up the problem solving on modern heterogeneous multiprocessor systems and multi-core processors. The paper extends the capabilities of existing blocked algorithms by utilizing blocks of unequal sizes. The extension supports natural graph modeling of real-world networks and allows the use of large sparse graphs divided into dense weakly connected clusters in the shortest path problems. Our approach is to compute shortest paths for all-pairs of cluster vertices represented by the cost adjacency matrix in advance, and afterward compute the shortest paths between vertices of different clusters through interconnect (bridge) edges in real time on demand. The approach is based on developing a new fast operation that accurately computes the shortest paths between vertices of one cluster that pass through the vertices of another neighboring cluster and through the edges connecting the clusters. Applying this operation to pairs of clusters allowed us to develop an approximate parallelizable algorithm, efficient regarding the CPU time and memory space consumed, that computes the shortest paths between the vertices within clusters and then between clusters. The algorithm can introduce inaccuracies in shortest paths when the weights of edges connecting clusters are small. It finds accurate solutions when the weights of these edges are larger than the weights of edges within clusters, e. g., in the case of road networks.

Keywords: sparse graph, cluster, shortest paths problem, blocked algorithm, unequally sized blocks, space and time efficiency.

For citation: Prihozhy A. A., Karasik O. N. Fast search for shortest paths in large sparse graphs divided into connected dense clusters. *Proceedings of BSTU, issue 3, Physics and Mathematics. Informatics*, 2024, no. 2 (284), pp. 96–103.
DOI: 10.52065/2520-6141-2024-284-13.

А. А. Прихожий[1], О. Н. Карасик[2]
[1]Белорусский национальный технический университет
[2] Иностранное производственное унитарное предприятие «Иссофт Солюшенз»

БЫСТРЫЙ ПОИСК КРАТЧАЙШИХ ПУТЕЙ В БОЛЬШИХ РАЗРЕЖЕННЫХ ГРАФАХ, РАЗДЕЛЕННЫХ НА СВЯЗНЫЕ ПЛОТНЫЕ КЛАСТЕРЫ

Целью статьи является решение задачи поиска кратчайших путей между всеми парами вершин в простых ориентированных взвешенных больших разреженных графах. Предполагается, что графы с положительными и отрицательными действительными весами ребер декомпозированы на слабосвязанные кластеры разного размера. Поскольку задача имеет множество практических приложений в различных областях, а графы могут быть слишком большими, наша цель – ускорить решение задачи на современных гетерогенных многопроцессорных системах и многоядерных процессорах. В статье расширяются возможности существующих блочных алгоритмов за счет использования блоков неравных размеров. Расширение поддерживает естественное и адекватное графовое моделирование реальных сетей и позволяет использовать большие разреженные графы, разделенные на плотные слабосвязанные кластеры, при решении задач о кратчайших путях. Наш подход заключается в том, чтобы заранее вычислять кратчайшие пути для всех пар вершин кластеров, представленных матрицей стоимости-смежности, а затем по запросу определять кратчайшие пути между вершинами разных кластеров в реальном времени. Подход основан на разработке новой быстрой операции, которая точно вычисляет кратчайшие пути между вершинами одного кластера, проходящие через вершины другого соседнего кластера и через ребра, соединяющие кластеры. Применение этой операции к парам кластеров позволило нам разработать приближенный распараллеливаемый алгоритм, эффективный по потребляемым процессорному

времени и объему памяти, вычисляющий кратчайшие пути внутри кластеров, а затем между кластерами. Алгоритм может вносить неточности в кратчайшие пути, когда веса ребер, соединяющих кластеры, малы. Он находит точные решения, когда веса этих ребер больше, чем веса ребер внутри кластеров, например, в случае дорожных сетей.

Ключевые слова: разреженный граф, кластер, задача о кратчайших путях, блочный алгоритм, блоки неравных размеров, пространственная и временная эффективность.

Для цитирования: Прихожий А. А., Карасик О. Н. Быстрый поиск кратчайших путей в больших разреженных графах, разделенных на связные плотные кластеры // Труды БГТУ. Сер. 3. Физико-математические науки и информатика. 2024. № 2 (284). С. 96–103 (На англ.).

DOI: 10.52065/2520-6141-2024-284-13.

Introduction. In this paper we consider the problem of finding shortest paths between all pairs of vertices in simple directed weighted large sparse graphs whose edges have positive and negative real weights, and which are decomposed into unequally sized clusters. Since the problem has numerous practical applications in various domains, our goal is to speed up the solution of the problem on the available resources of modern heterogeneous multi-processor systems and symmetric multicore processors.

Two main families of algorithms for solving the all-pairs shortest paths problem (*APSP*) are known: 1) those based on the Dijkstra-type *SSSP*-algorithm with a single-source (-sink) [1]; 2) those based on the Floyd-Warshall *APSP* algorithm [2]. Any algorithm from the first family must be applied to each source (sink) vertex. Each algorithm of the second family returns a complete solution to the problem.

The first family includes Dijkstra's algorithm assuming that the weights of the edges are positive, the Bellman-Ford and Johnson's algorithms assuming that the weights are positive and negative, and others [1, 3, 4]. The second family includes the Floyd-Warshall (*FW*) algorithm [2], the blocked Floyd-Warshall (*BFW*) algorithm proposed by Venkataraman, Park, Katz et al. [5–7], the graph extension-based algorithm *GEA* and the heterogeneous blocked *APSP* algorithm *HBAPSP* proposed by Prihozhy and Karasik in [8–11]. It is shown in [10–18] that blocked algorithms provide the following advantages: 1) suitability for parallelization; 2) locality of data references and efficient handling of CPU caches; 3) solving the graph scaling problem; 4) reduced power consumption; 5) use of GPUs; 6) use of dataflow networks of actors; 7) ability to handle unequally-sized blocks.

The blocked algorithms of the Floyd-Warshall family have computational complexity of $O(N^3)$ and space complexity of $O(N^2)$, where N is the number of vertices in graph. As N increases, the algorithms begin to consume huge amounts of CPU time and memory space. For example, if $N = 31{,}623$, then $N^3 = 31{,}623{,}446{,}801{,}367$ and $N^2 = 1{,}000{,}014{,}129$. A 4 GHz processor or core requires about $k1 \cdot 7906$ sec = $k1 \cdot 2.196$ hours of CPU time and about $k2 \cdot 1$ GB of RAM where $k1$ and $k2$ are factors. In many application domains the size of real graphs is much larger.

To solve the graph scaling problem, we develop an approach that decomposes a large sparse graph into dense parts (clusters) and sparse parts (interconnections of clusters). The former can be processed in the style of Floyd-Warshall family algorithms, and the latter can be processed in the style of Dijkstra family algorithms. When the size of the clusters is approximately the same, the spatial complexity of the dense parts can be reduced by the number, M of clusters. Thus, the algorithms we developed in this paper have a spatial complexity of <clusters_size> + <interconnections_size> that is <all_graph_size> / M + <interconnections_size> in the case where the clusters are of equal size. For sparse graphs, the <interconnections_size> is not large because it is only associated with bridge edges. Moreover, the CPU-time needed to execute *FW* on all clusters is M^2 times less than the CPU-time to execute *FW* on entire graph.

The main contribution of the paper is the development of a fast heterogeneous blocked approximate *APSP* algorithm for finding shortest paths in unequally sized clusters of large sparse directed graphs, as well as for finding shortest paths between vertices of different clusters in real time. The algorithm extends the heterogeneity of previously developed blocked algorithms with respect to both computation and allocation in memory of various types of blocks. It can give accurate solutions in such domains as road and other real-world networks.

Main part. The paper has the following structure: 1) introduction of all-pairs shortest paths algorithm on built unequal-size block matrices; 2) formulation of requirements for solving the shortest path problem on large sparse graphs partitioned into clusters; 3) development of an operation and algorithm for computing shortest paths between vertices of one cluster passing through vertices of other cluster and edges connecting the clusters; 4) development of fast algorithm calculating all-pairs shortest paths between vertices within clusters; 5) development of a fast algorithm computing shortest paths between vertices of different clusters in real time.

Blocked APSP algorithm working with blocks of unequal sizes. Let $G = (V, E)$ be a simple directed graph with real edge-weights consisting of a set V of vertices numbered from 1 to N and a set E of edges. Let W be a cost adjacency matrix for G. Let d_{ij} be a length of shortest path from vertex i to vertex j, and let D be a matrix of path distances for all pairs of vertices. The task of the *APSP* algorithm is to compute matrix D through matrix W.

In our recent work [11], we proposed to decompose a graph G into subgraphs (clusters) and to decompose the adjacency matrix into a matrix B of blocks of sizes defined by vector $S = (S_1 \ldots S_M)$. All diagonal blocks $B_{ii}[S_i \times S_i]$, $i = 1 \ldots M$ are square, and all non-diagonal blocks $B_{ij}[S_i \times S_j]$ are rectangular. We extended *BFW* to the all-pairs shortest path algorithm *APSPUS* (Algorithm 1) to handle unequally sized blocks. Its time complexity is N^3 (M^3 in terms of the number of blocks) and its spatial complexity is N^2 (M^2) because each block has the layout of tghe shortest path distance matrix (*DiM*).

Algorithm 1: Extension of *BFW* to account for the use of unequally sized blocks (*APSPUS*)

Input: A number N of graph vertices
Input: A cost adjacency matrix $W[N \times N]$
Input: A number M of blocks
Input: A vector $S = (S_1 \ldots S_M)$ of sizes of vertex subsets
Output: A blocked matrix $B[M \times M]$ of path distances

```
B[M × M] ← W[N × N]
for m ← 1 to M do
    BCUS(S, B, m, m, m)                    // D0
    for v ← 1 to M do
        if v ≠ m then
            BCUS(S, B, v, m, m)            // C1
            BCUS(S, B, m, m, v)            // C2
    for v ← 1 to M do
        if v ≠ m then
            for u ← 1 to M do
                if u ≠ m then
                    BCUS(S, B, v, m, u)    // P3
return B
```

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

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

```
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
return B
```

Algorithm 2 (*BCUS*) computes all blocks in Algorithm 1. *APSPUS* can be efficiently parallelized because all blocks of types *C*1 and *C*2 (and all blocks of type *P*3) can be executed in parallel.

Computation of shortest paths between vertices of one cluster through shortest paths of another cluster. In the case of matrix $B[2 \times 2]$, *APSPUS* computes two diagonal and two non-diagonal blocks (Fig. 1) in the following way ($\otimes$ is the MIN-PLUS matrix multiplication operation):

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

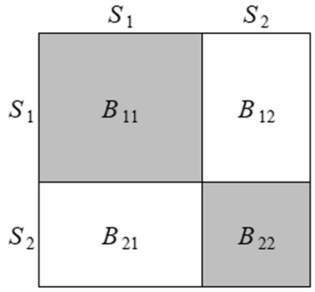

Fig. 1. Matrix $B[2 \times 2]$ in algorithm *APSPUS*

Equation (1) computes diagonal block B_{11} through itself. Equations (2)–(5) compute block B_{22} through block B_{11} and through itself. Equations (6)–(8) compute block B_{11} through block B_{22} and through itself. Each of the two intermediate blocks B_{21} and B_{12} is computed twice. For large sparse graphs, *APSPUS* requires huge memory space and processor time.

We propose a new exact method of computing B_{22} through B_{11} and computing B_{11} through B_{22}. Unlike *APSPUS*, the method allows to consider the features of sparse graphs with clustered vertices. It reduces the memory footprint and reduces the number of MIN-PLUS operations performed on blocks.

Let two clusters C_1 and C_2 partition the vertex set V of graph G into two subsets V_1 and V_2 of unequal sizes S_1 and S_2, respectively (Fig. 2). The clusters are represented by blocks B_{11} and B_{22} which first describe the weighted edges within the clusters, and then describe the lengths of shortest paths between vertices from set V_1 (block B_{11}) and between vertices from set V_2 (block B_{22}).

The sparse blocks W_{12} and W_{21} describe weighted bridge edges connecting vertices from V_1 to vertices from V_2 and vice versa, respectively. Blocks B_{11} and

B_{22} are placed in memory as distance matrices (*DiM*) using row-major memory layout, and blocks W_{12} and W_{21} are placed in memory as adjacent lists (*AjL*).

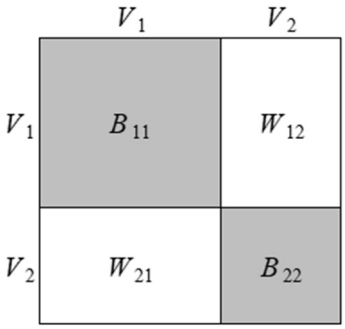

Fig. 2. Matrix $B[2 \times 2]$ in the new algorithm for calculating shortest paths in clusters represented by diagonal blocks

Our method of computing block B_{22} through block B_{11} and computing block B_{11} through block B_{22} is to perform the following five operations:

$B_{11} \leftarrow Diagonal\ (B_{11})$;
$B_{22} \leftarrow BlockTBlock\ (B_{22}, W_{21}, B_{11}, W_{12})$;
$B_{22} \leftarrow Diagonal\ (B_{22})$;
$B_{11} \leftarrow BlockTBlock\ (B_{11}, W_{12}, B_{22}, W_{21})$;
$B_{11} \leftarrow Diagonal\ (B_{11})$.

The $Diagonal(B_{ii})$ operation computes the shortest paths between all pairs of vertices of the set V_i corresponding to the block B_{ii}. The shortest paths can pass through both edges inside B_{ii} and edges outside B_{ii}. The operation can be implemented using *BCUS* or preferably using the faster *GEA* algorithm proposed in [8, 10].

The new operation $BlockTBlock(B_{jj}, W_{ji}, B_{ii}, W_{ij})$ computes the shortest paths between the vertices of block B_{jj} passing through the edges of block W_{ji}, then through the vertices and edges of block B_{ii} and finally through the edges of block W_{ij}. Since the edges of blocks W_{ji} and W_{ij} are not numerous in sparse graphs, the *BlockTBlock* operation is fast. If W_{ji} or W_{ij} are empty, the *BlockTBlock* operation is not performed at all. Moreover, the shortest paths of these blocks do not need to be stored in memory, only the edge descriptions need to be stored. This is a great advantage of our method, which is exact and gives accurate solutions.

Expanding the three operations $\otimes$ from (2)–(4) and using Algorithm 2, we derive Equation (9) which evaluates for each pair (i, j) of vertices of B_{22} the length of shortest path that passes through B_{11}. If the length is less than $B_{22}(i, j)$, the value of $B_{22}(i, j)$ is updated. Algorithm 3 is developed using (9). It is the first version of the *BCUS* compliant implementation of the *BlockTBlock* operation.

$$
\begin{aligned}
&B_{22}(i,j) = \\
&= min\ (B_{22}(i,j), \\
&\quad \min_{k \in V_1} (min(B_{21}(i,k), \\
&\qquad \min_{k_1 \in V_1} (B_{21}(i,k_1) + B_{11}(k_1,k))) + \\
&\quad + min\ (B_{12}(k,j), \\
&\qquad \min_{k_2 \in V_1} (B_{11}(k,k_2) + B_{12}(k_2,j))))).
\end{aligned} \tag{9}
$$

If the graph is sparse and the set of vertices is partitioned into clusters, Algorithm 3 executes redundant operations since it uses all vertices of sets V_1 and V_2 (although only bridge vertices can be used) and utilizes all elements of blocks W_{12} and W_{21} (although only bridge edges with non-infinite weights can be used). Moreover, the loop along k can also be considered redundant in the algorithm since the shortest paths in block B_{11} can be found by the *Diagonal* (B_{11}) function.

Algorithm 3: Calculation of diagonal block B_{22} through dense block B_{11} and sparse blocks W_{12}, W_{21} (*BlockTBlock, version 1*)

Input: Subsets V_1 and V_2 of vertex set V
Input: Blocks B_{11}, B_{22} and W_{12}, W_{21}
Output: Recalculated block B_{22}

```
for i ∈ V2 do
    for j ∈ V2 and j ≠ i do
        dij ← ∞
        for k ∈ V1 do
            dik ← ∞
            for k1 ∈ V1 do
                s1 ← W21(i, k1) + B11(k1, k)
                if dik > s1 then dik ← s1
            if dik > W21(i, k) then dik ← W21(i, k)
            dkj ← ∞
            for k2 ∈ V1 do
                s2 ← B11(k, k2) + W12(k2, j)
                if dkj > s2 then dkj ← s2
            if dkj > W12(k, j) then dkj ← W12(k, j)
            dikj ← dik + dkj
            if dij > dikj then dij ← dikj
        if B22(i, j) > dij then B22(i, j) ← dij
return B22
```

Let $V_{21} \subseteq V_2$ be a subset of the set V_2 of vertices that are bridges between clusters C_2 and C_1. Let E_{21} be a subset $E_{21} = \{(i, j) \mid i \in V_{21}$ and $j \in V_1\}$ of the set E of edges connecting clusters C_2 and C_1. These are called bridge edges. Similarly, let $V_{12} \subseteq V_1$ be the subset of bridge vertices between clusters C_1 and C_2, and $E_{12} = \{(i, j) \mid i \in V_{12}$ and $j \in V_2\}$ be the subset of bridge edges connecting clusters C_1 and C_2. We assume that the blocks B_{11}, B_{22} are represented as *DiM*, and the sets V_{21}, E_{12}, V_{12} and E_{21} are represented as *AjL*. Algorithm 4 describes a modified fast version 2 of the *BlockTBlock* operation, from which all redundant computations of Algorithm 3 have been removed.

Algorithm 4: Calculation of diagonal block B_{22} through diagonal block B_{11} and interconnect edges (*BlockTBlock, version* 2)

Input: Blocks B_{11}, B_{22}, W_{12} and W_{21}
Input: Subsets V_1 and V_2 of vertex set V
Input: Subsets $V_{12} \subseteq V_1$ and $V_{21} \subseteq V_2$ of bridge vertices
Input: Subsets $E_{12} \subseteq E$ and $V_{21} \subseteq E$ of bridge edges
Output: Recalculated block B_{22}
for $i \in V_{21}$ **do**
  for $(i, k1) \in E_{21}$ **and** $k1 \in V_1$ **do**
    for $k2 \in V_{12}$ **do**
      for $(k2, j) \in E_{12}$ **and** $j \in V_2$ **do**
        $w \leftarrow W_{21}(i, k1) + B_{11}(k1, k2) + W_{12}(k2, j)$
        if $B_{22}(i, j) > w$ **then**
          $B_{22}(i, j) \leftarrow w$
return B_{22}

Approximate algorithm for finding *APSPs in clusters of large sparse graphs (ACSG)*. Let the number of clusters satisfies $M > 2$. In this case we use the *BlockTBlock* algorithm by repeatedly computing diagonal blocks (clusters) through each other and through peripheral blocks. As a result, the shortest paths between pairs of vertices of each cluster are computed. Algorithm 5 performs two traversals through the clusters: forward and backward.

The *Diagonal**(S, B, P, d) function is executed $2 \cdot M - 1$ times and is realized using the *Diagonal* (B_{dd}) operation. The *BlockTBlock**(S, B, p, d) function is executed $M \cdot (M - 1)$ times and is realized using the *BlockTBlock* (B_{dd}, W_{dp}, B_{pp}, W_{pd}) operation. Thus, Algorithm 5 is very fast and and occupies little memory space (up to M times less than *APSPUS*). At the same time, it is approximate because it does not consider all paths passing through all clusters in any order.

Algorithm 5: Computing all-pairs shortest paths in clusters of sparse graphs (*ACSG*)

Input: A vector S of sizes of vertex subsets
Input: A blocked matrix $B[M \times M]$
Output: Recalculated matrix $B[M \times M]$
  for $d \leftarrow 1$ **to** M **do**
    *Diagonal**(S, B, d)
    if $d < M$ **then**
      for $p \leftarrow d + 1$ **to** M **do**
        *BlockTBlock**(S, B, p, d)
  for $d \leftarrow M$ **down to** 1 **do**
    if $d < M$ **then**
      *Diagonal**(S, B, d)
    if $d > 1$ **then**
      for $p \leftarrow d - 1$ **down to** 1 **do**
        *BlockTBlock**(S, B, p, d)
  return B

Fig. 3 shows the forward traversal and computation of clusters C_1 to C_3. Cluster C_1 is recomputed through itself, clusters C_2 and C_3 are recomputed through C_1 in parallel, cluster C_2 is recomputed through itself, and cluster C_3 is recomputed through C_2 and itself. Fig. 4 depicts the reverse traversal and computation of clusters from C_3 to C_1.

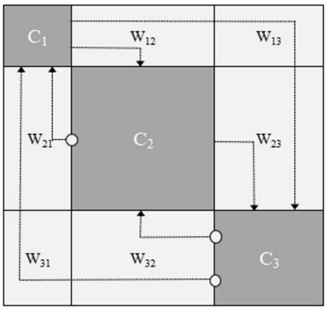

Fig. 3. Forward calculation of clusters from C_1 to C_3

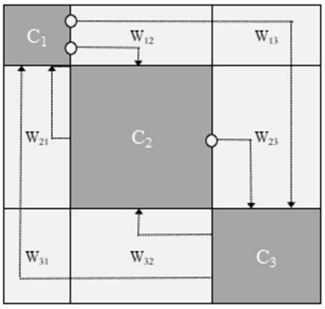

Fig. 4. Backward recalculation of clusters C_3 to C_1

Real-time computation of shortest paths between vertices of different clusters. An adaptation of the Dijkstra family algorithm provides accurate calculation of shortest paths between vertices of different clusters considering the shortest paths within clusters already accurately computed. The algorithm can be accelerated because the shortest paths between vertices within clusters have already been computed. The fast approximate Algorithm 6 can give good solutions if the graph satisfies the following constraints: 1) the clusters are connected to each other; 2) the weights of edges connecting clusters are greater than the weights of interior edges. For directly unconnected clusters it is not applicable.

Algorithm 6: Calculation of shortest paths between vertices of two clusters in real time (*ABCSG*)

Input: Blocks B_{11}, B_{22}, W_{12}
Input: Subsets V_1 and V_2 of vertex set V
Input: Subset $V_{12} \subseteq V_1$ of bridge vertices
Input: Subset $E_{12} \subseteq E$ of bridge edges
Output: Block B_{12}
for $i \in V_1$ **and** $j \in V_2$ **do** $B_{12}(i, j) \leftarrow \infty$
for $i \in V_1$ **do**
  for $i1 \in V_{12}$ **do**
    for $(i1, j1) \in E_{12}$ **and** $j1 \in V_2$ **do**
      for $j \in V_2$ **do**
        $w \leftarrow B_{11}(i, i1) + W_{12}(i1, j1) + B_{22}(j1, j)$
        if $B_{12}(i, j) > w$ **then**
          $B_{12}(i, j) \leftarrow w$
return B_{12}

Results. The proposed algorithms were implemented in C++ language using Visual Studio 2022 under OS Windows 10. Experimental results were obtained on Intel Core i7-10700 CPU processor. Fig. 5 shows the speedup of 22.8 to 28.4 times given by the proposed all-pairs shortest paths approximate *ACSG* and *ABCSG* algorithms compared to the exact *APSPUS* algorithm on random graphs of 1,200, 2,400, 3,600 and 4,800 vertices. All graphs consist of 10 clusters of different sizes and with different numbers of edges. The price of this high speedup is the small inaccuracies in the calculation of shortest paths introduced by the approximate algorithms.

We evaluated the inaccuracies in shortest paths within and between clusters of the graphs consisting of 2,400 vertices and 139,444 edges with different numbers of bridge edges and different bridge edge weights. Fig. 6 shows the dependence of inaccuracies (%) in shortest paths lengths within clusters on the average bridge edge weights varying from 0 to 40 with an average edge weight of 54.55 within clusters. In Fig. 6, solid, dashed, and dotted lines correspond to 1,177, 5,761 and 11,478 bridge edges, respectively. It can be observed that the inaccuracies are higher for graphs with a larger number of bridge edges and with smaller bridge edge weights. When the bridge edge weight is greater than 40, ACSG becomes an accurate algorithm. The inaccuracy is only 0.031% for 1,177 bridge edges with weight 10, is 0.012% for 5,761 edges with weight 20, and is 0.049% for 11,478 edges with weight 20. All weights are less than the average weight of 54.55 for edge within clusters.

Fig. 7 shows the effect of bridge edge weights on the inaccuracies (%) given by *ABCSG* when computing the lengths of shortest paths between clusters of the sparse graphs with the same parameters. A graph with 1,177 edges connecting clusters gave inaccuracies ranging from 23.7% down to 0% when the bridge edge weight varied from 0 to 99 (0.014% for a weight of 50). When the graph has 5,761 bridge edges, the inaccuracies range from 163.6% down to 0% when the bridge edge weights range from 0 to 99 (0.018% for a weight of 50). When the graph has 11,478 bridge edges, the inaccuracies are from 100.9% down to 0% for a bridge edge weight of 0 to 40 (0.035% for a weight of 50).

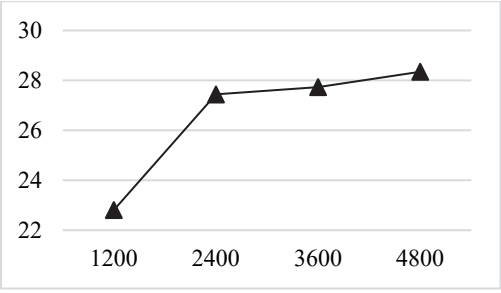

Fig. 5. Speedup (times) of *ACSG-ABCSG* compared to *APSPUS* vs. graph size

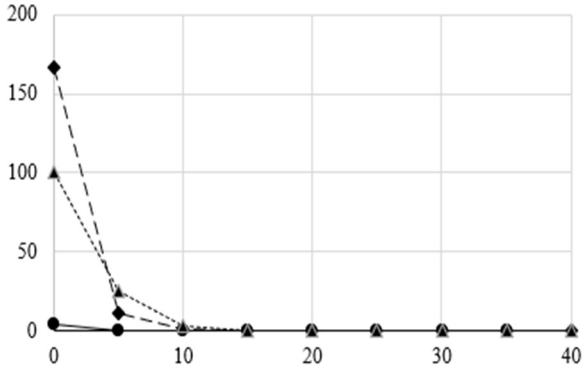

Fig. 6. Inaccuracies (%) in shortest paths lengths within clusters *ACSG* has given for graphs of 2,400 vertices vs. bridge edge weight

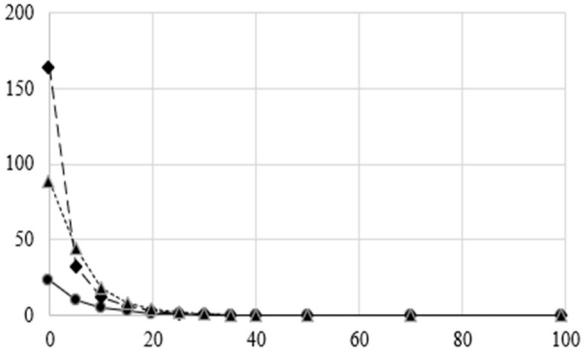

Fig. 7. Inaccuracies (%) in shortest paths lengths between clusters the *ABCSG* has given for graphs of 2,400 vertices vs. bridge edge weight

Conclusion. In the paper, we have developed an approach for solving the all-pairs shortest paths problem on large sparse graphs partitioned into dense weakly connected clusters. The key advantages of the approach are the reduction of the memory footprint and the reduction of the CPU time consumed. The approach is based on our recently published blocked algorithms that operate on unequally sized blocks of the cost adjacency matrix. In this paper, we proposed a very fast exact algorithm that implements an operation of computing the shortest paths between vertices of one cluster passing through vertices of a neighboring cluster, and through edges connecting the clusters. This operation is the basis of a time- and memory-efficient approximate algorithm that computes the shortest paths within all clusters of the graph. The shortest paths between vertices of different clusters are computed in real time. The algorithm is up to 28 times faster than the blocked Floyd-Warshall family algorithm. It can provide accurate solutions for roads and other networks since inaccuracies in shortest paths are negligible when the weights of bridge edges are greater than the weights of interior edges of 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. Venkataraman G., Sahni S., Mukhopadhyaya S. A Blocked All-Pairs Shortest Paths Algorithm. *Journal of Experimental Algorithmics* (JEA), 2003, vol. 8, pp. 857–874.
6. 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.
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. Prihozhy A. A., Karasik O. N. Heterogeneous blocked all-pairs shortest paths algorithm. *Sistemnyy Analiz i prikladnaya informatika* [System analysis and applied information science], 2017, no. 3, pp. 68–75. (In Russian).
9. Prihozhy A. A., Karasik O. N. Advanced heterogeneous block-parallel all-pairs shortest path algorithm. *Trudy BGTU* Proceedings of BSTU, issue 3, Physics and Mathematics. Informatics, 2023, no. 1 (266), pp. 77–83.
10. 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.
11. Prihozhy A., Karasik O. New blocked all-pairs shortest paths algorithms operating on blocks of unequal sizes. *System analysis and applied information science*, 2023, no. 4, pp. 4–13.
12. 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.
13. Yang S. Liu X., Wang Y., He X., Tan G. Fast All-Pairs Shortest Paths Algorithm in Large Sparse Graph. *ICS'23 Proceedings of 37th International conference on Supercomputing*, 2023, pp. 277–288.
14. 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.
15. 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.
16. Prihozhy A. A., Karasik O. N. Influence of shortest path algorithms on energy consumption of multicore processors. *System analysis and applied information science*, 2023, no. 2, pp. 4–12.
17. 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.
18. Prihozhy A., Merdjani R., Iskandar F. Automatic Parallelization of Net Algorithms. *Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC'00)*. Quebec, Canada, 2000, pp. 24–28.

Список литературы

1. Dijkstra E. W. A note on two problems in connexion with graphs // Numerische Mathematik. 1959. Vol. 1, no. 1. P. 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. P. 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. Venkataraman G., Sahni S., Mukhopadhyaya S. A Blocked All-Pairs Shortest Paths Algorithm. Journal of Experimental Algorithmics (JEA). 2003. Vol. 8. P. 857–874.
6. Park J. S., Penner M., Prasanna V. K. Optimizing graph algorithms for improved cache performance // IEEE Trans. on Parallel and Distributed Systems. 2004. No. 15 (9). P. 769–782.
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. 2008. P. 47–55.
8. Прихожий А. А., Карасик О. Н. Разнородный блочный алгоритм поиска кратчайших путей между всеми парами вершин графа // Системный анализ и прикладная информатика. 2017. № 3. С. 68–75.

9. Prihozhy A. A., Karasik O. N. Advanced heterogeneous block-parallel all-pairs shortest path algorithm // Труды БГТУ. Сер. 3, Физико-математические науки и информатика. 2023. № 1 (266). С. 77–83.

10. Prihozhy A. A., Karasik O. N. Influence of shortest path algorithms on energy consumption of multicore processors // System analysis and applied information science. 2023. No. 2. P. 4–12.

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

12. 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. P. 360–369.

13. Yang S. Liu X., Wang Y. He X., Tan G. Fast All-Pairs Shortest Paths Algorithm in Large Sparse Graph // ICS'23 Proceedings of 37th International conference on supercomputing. 2023. P. 277–288.

14. Prihozhy A. A. Optimization of data allocation in hierarchical memory for blocked shortest paths algorithms // System analysis and applied information science. 2021. No. 3. P. 40–50.

15. 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. P. 57–65.

16. 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. P. 56–66.

17. Prihozhy A. A. Generation of shortest path search dataflow networks of actors for parallel multicore implementation // Informatics. 2023. Vol. 20, no. 2. P. 65−84.

18. Prihozhy A., Merdjani R., Iskandar F. Automatic Parallelization of Net Algorithms // Proceedings of the International conference on parallel computing in electrical engineering (PARELEC'00). Quebec, Canada, 2000. P. 24–28.

Information about the authors

Prihozhy Anatoly Alexievich − DSc (Engineering), Professor, Professor, the Department of Computer and System Software. Belarusian National Technical University (65, Nezalezhnasti Ave., 220013, Minsk, Republic of Belarus). E-mail: prihozhy@yahoo.com

Karasik Oleg Nikolayevich − PhD (Engineering), Lead Engineer, ISsoft Solutions (5, Chapaeva str., 220034, Minsk, Republic of Belarus). E-mail: karasik.oleg.nikolaevich@gmail.com

Информация об авторах

Прихожий Анатолий Алексеевич − доктор технических наук, профессор, профессор кафедры программного обеспечения информационных систем и технологий. Белорусский национальный технический университет (220013, г. Минск, пр. Независимости 65, Республика Беларусь). E-mail: prihozhy@yahoo.com

Карасик Олег Николаевич − кандидат технических наук, ведущий инженер. Иностранное производственное унитарное предприятие «Иссофт Солюшенз». (220034, г. Минск, ул. Чапаева 5, Республика Беларусь). E-mail: karasik.oleg.nikolaevich@gmail.com

Received 15.03.2024