Теоретические основы информатики

Theoretical foundations of computer science

УДК 004

РАЗНОРОДНЫЙ БЛОЧНЫЙ АЛГОРИТМ ПОИСКА КРАТЧАЙШИХ ПУТЕЙ МЕЖДУ ВСЕМИ ПАРАМИ ВЕРШИН КЛАСТЕРИЗОВАННОГО ВЗВЕШЕННОГО ГРАФА

А. А. ПРИХОЖИЙ[1)], ***О. Н. КАРАСИК***[2)]

[1)]*Белорусский национальный технический университет, пр. Независимости, 65, 220013, г. Минск, Беларусь*
[2)]*Иссофт Солюшенз, ул. Чапаева, 5, 220034, г. Минск, Беларусь*

Аннотация. Предлагается новый гетерогенный блочный алгоритм поиска кратчайших путей между всеми парами вершин большого ориентированного взвешенного простого графа, состоящего из слабосвязанных плотных кластеров (подграфов) разных размеров. Алгоритм учитывает и активно использует входные и выходные граничные вершины и ребра каждого кластера для ускорения вычислений и локализации обращений к памяти. Он делит все блоки матрицы «стоимость – смежность» на четыре типа (квадратный диагональный, прямоугольный вертикальный на кресте, прямоугольный горизонтальный на кресте и прямоугольный периферийный) и использует отдельную процедуру расчета для них, учитывает конструктивные особенности самого блока и способ его расчета через другие блоки. Приводится теоретическое обоснование преимуществ предлагаемых алгоритмов, сокращающих

Образец цитирования:
Прихожий АА, Карасик ОН. Разнородный блочный алгоритм поиска кратчайших путей между всеми парами вершин кластеризованного взвешенного графа. *Журнал Белорусского государственного университета. Математика. Информатика.* 2025;3:62–75 (на англ.).
EDN: SJMQEH

For citation:
Prihozhy AA, Karasik ON. Heterogeneous blocked all-pairs shortest paths algorithm for clustered weighted graphs. *Journal of the Belarusian State University. Mathematics and Informatics.* 2025;3:62–75.
EDN: SJMQEH

Авторы:
Анатолий Алексеевич Прихожий – доктор технических наук, профессор; профессор кафедры программного обеспечения информационных систем и технологий факультета информационных технологий и робототехники.
Олег Николаевич Карасик – кандидат технических наук; ведущий инженер.

Authors:
Anatoly A. Prihozhy, doctor of science (engineering), full professor; professor at the department of software of information systems and technologies, faculty of information technology and robotics.
prihozhy@yahoo.com
https://orcid.org/0000-0002-1941-0806
Oleg N. Karasik, PhD (engineering); leading engineer.
karasik.oleg.nikolaevich@gmail.com

время выполнения при поиске кратчайших путей. Достоверность сформулированных положений подтверждается результатами проведенных вычислительных экспериментов. Разрабатываются однопоточные реализации и многопоточные OpenMP реализации предлагаемого гетерогенного алгоритма и двух известных гомогенных блочных алгоритмов для поиска кратчайших путей. Вычислительные эксперименты на многоядерных процессорах проводятся на случайных ориентированных взвешенных графах, декомпозированных на слабосвязанные плотные кластеры разных размеров. Описываются результаты для четырех кластеризованных графов, два из которых имеют 4800 вершин (20 и 41 кластер соответственно) и два из которых имеют 9600 вершин (40 и 80 кластеров соответственно). На компьютере MacBook M1 Max в случае с однопоточностью предложенный гетерогенный блочный алгоритм для кластеризованных графов с граничными вершинами превзошел известный гомогенный блочный алгоритм для таких же графов в 1,62–1,94 раза; в случае с OpenMP-многопоточностью ускорение составило 1,87–1,97. На сервере из двух процессоров Intel Xeon E5-2620v4 гетерогенный алгоритм превзошел гомогенный алгоритм в 1,58–1,66 раза для однопоточности и в 1,29–1,64 раза для многопоточности. Сравнение предложенного алгоритма с классическим блочным алгоритмом Флойда – Уоршелла, в котором блоки имеют одинаковый размер, показало ускорение в 4,17–8,18 раза в случае с однопоточностью и ускорение в 3,91–6,36 раза в случае с OpenMP-многопоточностью.

Ключевые слова: кластеризованный взвешенный большой граф; кратчайшие пути между всеми парами вершин; блочный алгоритм; гетерогенные вычисления; ускорение.

HETEROGENEOUS BLOCKED ALL-PAIRS SHORTEST PATHS ALGORITHM FOR CLUSTERED WEIGHTED GRAPHS

A. A. PRIHOZHY[a]***, O. N. KARASIK***[b]

[a]*Belarusian National Technical University, 65 Niezaliezhnasci Avenue, Minsk 220013, Belarus*
[b]*ISsoft Solutions, 5 Chapajeva Street, Minsk 220034, Belarus*
Corresponding author: A. A. Prihozhy (prihozhy@yahoo.com)

Abstract. New heterogeneous blocked algorithm of finding all-pairs shortest paths in a large directed weighted simple graph consisting of weakly connected dense clusters (subgraphs) of different sizes is proposed. The algorithm considers and actively exploits the input and output bridge-vertices and edges of each cluster to speed up computation and localise memory accesses. It divides all blocks of the cost adjacent matrix into four types (square diagonal, rectangular vertical cross, rectangular horizontal cross and rectangular peripheral) and uses a separate computation procedure for each type, considering the design features of the block itself and the way it is computed through other blocks. A theoretical justification of the advantages of the proposed algorithms, which reduce the execution time when searching for the shortest paths, is given. The validity of the formulated statements is also confirmed by the results of computational experiments. We have developed single-threaded implementations and multi-threaded OpenMP implementations of the proposed heterogeneous algorithm and two known homogeneous blocked algorithms of finding shortest paths. Computational experiments on multicore processors were performed on directed weighted random sparse graphs decomposed into weakly connected dense clusters of different sizes. The results are described for four clustered graphs, two of which have 4800 vertices (20 and 41 clusters, respectively) and two of which have 9600 vertices (40 and 80 clusters, respectively). On the MacBook M1 Max computer in the case of single-threaded implementations proposed heterogeneous blocked algorithm for clustered graph with bridge-vertices outperformed the known homogeneous blocked algorithm for the same graphs by a factor of 1.62–1.94; in the case of multi-threaded OpenMP implementations the speedup was 1.87–1.97. On a server with Intel Xeon E5-2620v4 processors heterogeneous algorithm outperformed the known homogeneous algorithm by a factor of 1.58–1.66 for single-threaded implementations and by a factor of 1.29–1.64 for multi-threaded implementations. A comparison of proposed algorithm with the classical blocked Floyd – Warshall algorithm in which all blocks are of the same size showed a speedup of 4.17–8.18 times in the case of single-threaded implementations and a speedup of 3.91–6.36 times in the case of multi-threaded OpenMP implementations.

Keywords: clustered weighted large graph; all-pairs shortest paths; blocked algorithm; heterogeneous computations; speedup.

Introduction

The problem of all-pairs shortest paths (APSP) is fundamental in many domains including social networks, bioinformatics, transportation networks, synthesis of quantum logic circuits, etc. [1; 2]. The classical Floyd – Warshall algorithm (further *FW*) [3; 4] solves this problem. The blocked *FW* (further *BFW*) [5–7], which is

homogeneous, performs a partitioning of the graph into subgraphs of equal size and uses the same block calculation procedure for all blocks of the distance matrix. New APSP algorithms for large clustered graphs were proposed in the works [8; 9]. They combine the classical *FW* and Dijkstra algorithm and utilise bridge-vertices of the clusters to improve performance. The heterogeneous blocked APSP algorithm [10; 11] for non-clustered dense graphs distinguishes four types of blocks, each computed by a separate procedure, but all blocks are of the same size. The homogeneous *BFW*, which can handle subgraphs and blocks of different sizes [12; 13], uses the same universal procedure to calculate the blocks of all types. The blocked APSP algorithm for clustered graphs and unequally sized blocks [14], which uses bridge-vertices to reduce runtime, is homogeneous because it uses a single block calculation procedure. The works [15–22] improve *BFW* while considering aspects such as efficient utilisation of graphics processing units, tuning to optimal block size, using cooperative thread scheduler, reducing power consumption, establishing dataflow networks of actors, etc. In this paper, we present a new heterogeneous blocked algorithm for solving the problem of APSP on large clustered graphs considering bridge-vertices and bridge-edges, unequally sized blocks, separate computation procedures for each block type and algorithm transformation for localising data references.

Homogeneous *BFW*

Let $G = (V, E)$ be a directed simple graph with real edge-weights composed of a set V of N vertices and a set E of edges. A cost adjacency matrix W of G has $w_{i,i} = 0$, $1 \le i \le N$, where $w_{i,i}$ is equal to the weight of edge $(i, j) \in E$, and $w_{i,j} = \infty$ if $i \ne j$ and $(i, j) \notin E$. When G has no negative-weight cycle, the dynamic programming *FW* [3; 4] computes a sequence of distance matrices $D^0 = W, \ldots, D^k, \ldots, D^N$ such that in matrix D^k the shortest path from i to j is composed of the vertex subset $\{1, \ldots, k\}$. *FW* calculates the elements of matrix D using the formula

$$d_{i,j}^k = \min\left(d_{i,j}^{k-1}, d_{i,k}^{k'} + d_{k,j}^{k''}\right) \tag{1}$$

and assuming that $d_{i,j}^0 = w_{i,j}$.

Claim 1 [6]. We suppose that $d_{i,j}^k$, $k = 1, \ldots, N$, is computed with the formula (1) for k', $k'' \ge k - 1$ and k', $k'' \le N$, then upon termination *FW* correctly computes APSP.

Algorithm 1 (homogeneous *BFW*) [5; 6; 10–14] divides set V into subsets of equal size S and splits the matrix D into blocks leading to matrix $B[M \times M]$. Initially each block has a zero level of calculation, $B^0[v, u] = W[v, u]$.

Algorithm 1. Homogeneous *BFW*
for $m \leftarrow 1, \ldots, M$ **do**
  $B^m[m, m] \leftarrow BCal\left(B^{m-1}[m, m], B^{m-1}[m, m], B^{m-1}[m, m]\right)$ // $D0$ block
  for $v \in \{1, \ldots, M\}$ **and** $v \ne m$ **do**
    $B^m[v, m] \leftarrow BCal\left(B^{m-1}[v, m], B^{m-1}[v, m], B^m[m, m]\right)$ // $C1$ block
    $B^m[m, v] \leftarrow BCal\left(B^{m-1}[m, v], B^m[m, m], B^{m-1}[m, v]\right)$ // $C2$ block
  for $v, u \in \{1, \ldots, M\}$ **and** $v \ne m$ **and** $u \ne m$ **do**
    $B^m[v, u] \leftarrow BCal\left(B^{m-1}[v, u], B^m[v, m], B^m[m, u]\right)$ // $P3$ block
return B^M.

In each of M iterations of the loop along m, homogeneous *BFW* computes one diagonal $D0$ block $B^m[m, m]$ to level m of calculation, computes $M - 1$ vertical $C1$ cross-blocks $B^m[v, m]$ to level m, computes $M - 1$ horizontal $C2$ cross-blocks $B^m[m, v]$ to level m and computes $(M-1)^2$ peripheral $P3$ blocks $B^m[v, u]$ to level m. All cross-blocks can be calculated mutually in parallel. All peripheral blocks can be calculated in parallel as well. The following claim holds for all the blocks of type $P3$ [19; 20].

Claim 2. We suppose that the $P3$ block $B^m[v, u]$, $m = 1, \ldots, M$, is computed with the formula

$$B^m[v, u] = BCal\left(B^{m-1}[v, u], B^{m'}[v, m], B^{m''}[m, u]\right)$$

for m', $m'' \ge m$ and m', $m'' \le M$. Upon termination *BFW* correctly computes APSP in graph G.

At each iteration *BFW* increments the calculation level of each block and fulfills the requirement of claim 2 for the $P3$ blocks. Therefore, *BFW* computes APSP correctly.

The block calculation algorithm 2 (*BCal*) implements the classical *FW* and can compute *D*0, *C*1, *C*2 and *P*3 blocks in three loops along *k*, *i* and *j* (loops along *i* and *j* can be reordered). Its three input blocks are denoted $B1^0$, $B2^0$ and $B3^0$, respectively. Every execution of the assignment in the nest of loops increases the calculation level of element $B1_{ij}$ from $k-1$ to k. The assignment fulfills the requirement of claim 1; therefore, *BCal* correctly computes *B*1 block over itself, *B*2 and *B*3 blocks. But if we reorder the loop along *k* with the loop along *i* or *j*, the requirement is not fulfilled, and the algorithm becomes incorrect.

Algorithm 2. *BCal* (single calculation of all blocks)
for $k \leftarrow 1, \ldots, S$ **do**
  for $i, j \in \{1, \ldots, S\}$ **do**
    $B1_{i,j}^{k} \leftarrow \min\left(B1_{i,j}^{k-1}, B2_{i,k}^{k'} + B3_{k,j}^{k''}\right)$
return *B*, *P*.

Directed weighted sparse graphs consisting of unequally sized clusters

In the paper we call a graph clustered if it can be partitioned into dense subgraphs that are interconnected by a few edges. We call a subgraph dense if its density (the actual number divided by the maximum number of directed edges) is over 0.5. We call a graph sparse if its density is less than 0.3. Figure 1 shows an example directed clustered graph that consists of 17 vertices, 69 edges and 3 clusters including 5, 5 and 7 inner vertices, respectively, and 14, 15 and 31 one-direction-edges, respectively. The clusters have densities of 0.70, 0.75 and 0.74, respectively, and are interconnected by 9 one-direction-edges. The density of the entire graph is 0.25. Let *C* be the set of all clusters, $C[c].\text{vert}$ be the set of vertices of cluster *c* and $C[c].\text{size}$ or $\text{size}(c)$ be the vertex set size. All vertices are numbered within the cluster, so $C[c].\text{vert}[v].\text{index} \in \{1, \ldots, C[c].\text{size}\}$ is assigned to each $v \in C[c].\text{vert}$.

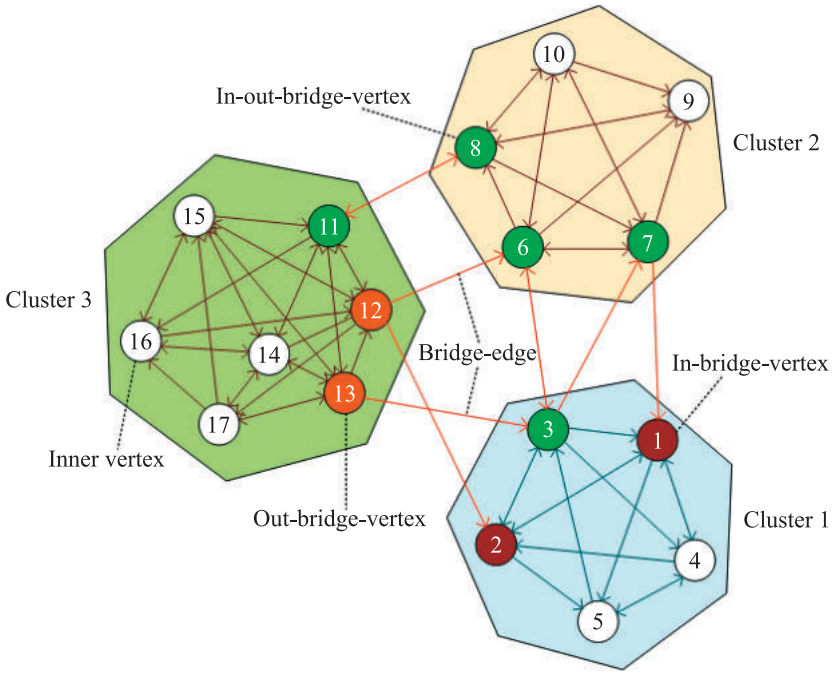

Fig. 1. An example graph consisting of weakly connected clusters

The algorithms we have developed exploit the concept of cluster bridge-vertices. A vertex is a bridge in cluster $c \in C$ if it is incident to an edge, which connects the vertex with a vertex of other cluster $e \in C \setminus \{c\}$. Let $C[c].\text{bridge}$ be the set of bridge-vertices of cluster *c* such as $C[c].\text{bridge} \subseteq C[c].\text{vert}$. A vertex *v* is an input bridge in cluster *c* if it has an incoming incident edge connecting *v* with other clusters' vertices. Let $C[c].\text{bridge.inall}$ be the set of input bridge-vertices of cluster *c* such as $C[c].\text{bridge.inall} \subseteq C[c].\text{bridge}$. A vertex *v* is an output bridge in cluster *c* if it has an outgoing incident edge connecting *v* with other clusters' vertices. Let

$C[c]$.bridge.outall be the set of output bridge-vertices of cluster c such as $C[c]$.bridge.outall $\subseteq C[c]$.bridge. The set $C[c]$.bridge.inout includes the vertices of cluster c, which are input and output bridges simultaneously: $C[c]$.bridge.inout $= C[c]$.bridge.inall $\cap\, C[c]$.bridge.outall. The set $C[c]$.bridge.in includes the purely input bridge-vertices of cluster c: $C[c]$.bridge.in $= C[c]$.bridge.inall $\setminus [c]$.bridge.outall. The set $C[c]$.bridge.out includes purely output bridge-vertices of c: $C[c]$.bridge.out $= [c]$.bridge.outall $\setminus\, C[c]$.bridge.inall. In fig. 1 cluster 1 has two input bridges (vertices 1 and 2) and one output bridge (vertex 3), vertices 4 and 5 are inner. Cluster 2 has three input and output bridges (vertices 6, 7 and 8), vertices 9 and 10 are inner. Cluster 3 has one input and output bridge (vertex 10) and two output bridges (vertices 11 and 12), vertices 13–16 are inner.

Having a partitioning C of graph G into clusters, we create matrix B of blocks of the shortest path distances between the vertices (fig. 2). A block $B[m, m]$ (below denoted $B[m]$) of dimension $C[m]$.size $\times\, C[m]$.size lies on the principal diagonal of matrix B and describes the shortest path distances between the vertices of cluster m. A block $B[c, e]$ of dimension $C[c]$.size $\times\, C[e]$.size lies out of the principal diagonal and describes the shortest path distance between the vertices of cluster c and the vertices of cluster e.

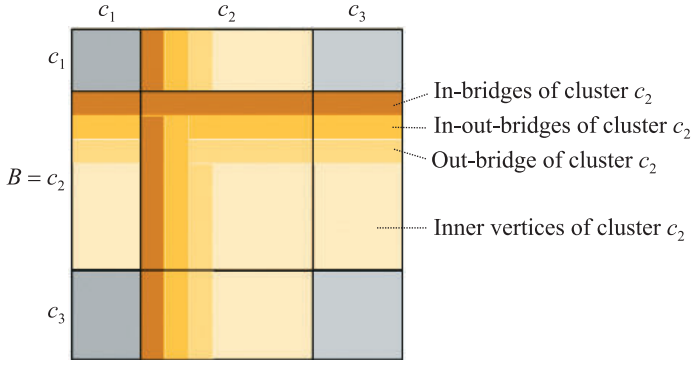

Fig. 2. Layout of distance-adjacency matrix in memory

A block element $B[c, e]_{i,j}$ is identified with indices $i \in \{1, \ldots, C[c].\text{size}\}$ and $j = \{1, \ldots, C[e].\text{size}\}$ of vertices in clusters c and e, respectively. Within each block the vertices are placed in the following order: input bridges, input and output bridges, output bridges and other inner vertices of the cluster. This allows us to localise and assign the vertices of the same group in the same memory lines, and to speed up computations with the algorithms we propose.

Heterogeneous blocked APSP algorithm for clustered graphs

In developing the heterogeneous blocked APSP algorithm for clustered graphs upon unequal block sizes and bridge-vertices (algorithm 3 (*HBSPCG*)) we combined the following techniques: handling blocks of unequal sizes, using input and output bridge-vertices of clusters to speed up computation, using a heterogeneous approach to compute four types of blocks and considering all the features of each block type. Its input is a matrix W describing the graph. Its output is matrix B of the shortest path distances between all pairs of vertices. Algorithm 3 describes *HBSPCG* at the level of block computation sub-algorithms. Algorithm 4 (*D*0*CG*) calculates the shortest path segments between the vertices of one subgraph, algorithm 7 (*C*1*CG*) and algorithm 8 (*C*2*CG*) – between the vertices of two subgraphs, and algorithm 9 (*P*3*CG*) – between the vertices of two subgraphs, provided that the path passes through the vertices of the third subgraph. *HBSPCG* organises the correct recalculation of the shortest path segments, when searching for APSP in the entire graph. At the block level the same control flow scheme is used as in *BFW*. The key difference between the algorithms is that *HBSPCG* uses four separate sub-algorithms for calculating *D*0, *C*1, *C*2 and *P*3 blocks with different parameter profiles. The structure of matrix B in *HBSPCG* is different from that in *BFW*.

Algorithm 3. *HBSPCG*
$B^0[M \times M] \leftarrow W[N \times N]$
for $m \leftarrow 1, \ldots, M$ **do**
  $B^m[m] \leftarrow D0CG\left(C, m, B^{m-1}[m]\right)$ // *D*0 block
  for $c \in \{1, \ldots, M\}$ **and** $v \neq m$ **do**

$B^m[c, m] \leftarrow C1CG\left(B^{m-1}[c, m], B^m[m], C, c, m\right)$ // $C1$ block

$B^m[m, c] \leftarrow C2CG\left(B^{m-1}[m, c], B^m[m], C, m, c\right)$ // $C2$ block

for $c, e \in \{1, \ldots, M\}$ **and** $c \neq m$ **and** $e \neq m$ **do**

$B^m[c, e] \leftarrow P3CG\left(B^{m-1}[c, e], B^m[c, m], B^m[m, e], C, c, m, e\right)$ // $P3$ block

return B^M.

At the control flow level the correctness of *HBSPCG* is ensured in the same way as *BFW*. The computation level of each block, regardless of type, increases at each iteration of the loop along m. The main difference between *HBSPCG* and homogeneous *BFW* is that the former is aimed at saving central processing unit and memory resources, when calculating diagonal, vertical, horizontal and peripheral blocks.

New algorithm for calculating diagonal blocks

All blocks are square on the principal diagonal of B. When recalculating a diagonal block over itself, the bridge-vertices are not considered. *D*0*CG*, which we propose as a sub-algorithm of *HBSPCG*, uses one cluster m and computes one square block $B[m, m]^0$ (which will also be denoted $B[m]^0$) of dimension $C[m].\text{size} \times C[m].\text{size}$ from computation level 0 to S. Since the block is computed over itself, the order of computing the elements $B[m]^k_{i, j}$ through elements $B[m]^{k'}_{i, k}$ and $B[m]^{k''}_{k, j}$ along k is crucial. It must satisfy the formula (1). The block describes APSP between the vertices of cluster m, which may pass through the vertices of other clusters. We interpret block $B^0[m]$ as a matrix of edge weights between the vertices of cluster m.

*D*0*CG* consists of two nests of loops. The first nest includes three loops along variables k, i and j and three assignments with the min operation. The first two loops along i and j cover $k-1$ vertex of the subgraph of cluster m. The loop along k is repeated over the indices of vertices $C[m].\text{vert}$ of cluster m. In *D*0*CG* three assignments are aimed at updating APSP between vertices i and j, between vertices i and k, and between vertices k and j over new paths of shorter length. The second nest consists of two loops along i and j. Finally, it computes the shortest path between vertices $1, \ldots, S-1$ through the row and column labeled by S in the matrix $B[m]$. We developed *D*0*CG* as a competing alternative to *BCal*.

Algorithm 4. *D*0*CG* (calculation of the diagonal square block $B[m]$)

$S \leftarrow C[m].\text{size}$

for $k \leftarrow 2, \ldots, S$ **do**

  for $i \leftarrow 1, \ldots, k-1$ **do**

    for $j \leftarrow 1, \ldots, k-1$ **do**

      $B[m]^{k-1}_{i, j} \leftarrow \min\left(B[m]^{k-2}_{i, j}, B[m]^{k-1}_{i, k-1} + B[m]^{k-1}_{k-1, j}\right)$ // Ω_{k-1}

      $B[m]^{j+1}_{i, k} \leftarrow \min\left(B[m]^{j}_{i, k}, B[m]^{k-1}_{i, j} + B[m]^{0}_{j, k}\right)$ // Λ_k

      $B[m]^{i+1}_{k, j} \leftarrow \min\left(B[m]^{i}_{k, j}, B[m]^{0}_{k, i} + B[m]^{k-1}_{i, j}\right)$ // Λ_k

for $i \leftarrow 1, \ldots, S-1$ **do**

  for $j \leftarrow 1, \ldots, S-1$ **do**

    $B[m]^{S}_{i, j} \leftarrow \min\left(B[m]^{S-1}_{i, j}, B[m]^{S}_{i, S} + B[m]^{S}_{S, j}\right)$ // Ω_S

return $B[m]^S$.

Theorem 1. *Upon termination D*0*CG correctly computes the diagonal block* $B[m]^S$ *over* $B[m]^0$*, which represents the shortest path lengths between all vertices of cluster m, possibly passing through vertices of other clusters of graph G.*

P r o o f. *D*0*CG* does not consider the bridge-vertices of cluster m because it computes the block $B[m]$ over itself. The competitive algorithm *BCal* recomputes each element of the block $B[m]$ at each iteration of the loop along k. Unlike *BCal*, *D*0*CG* starts with a one-vertex graph and a block $B[m]^1$ of dimension 1×1. It then iteratively adds one row k and one column k to block $B[m]^{k-1}$ and obtains block $B[m]^k$. The procedure is

illustrated in fig. 3, *a*. In this figure $B[m]^0$ denotes input block $B[m]$ of dimension $S \times S$ before executing *D*0*CG*. Variable b_{ij} denotes an element of the matrix $B[m]$.

Two operations Λ_k and Ω_k are used to accomplish the procedure. The first operation Λ_k computes APSP that are represented by column k and row k of block $B[m]^k$ and are computed over APSP of subblock $B[m]^{k-1}$. Column k is computes by the following equation:

$$B[m]_{ik}^{j+1} = \min\left(B[m]_{ik}^{j},\ B[m]_{ij}^{k-1} + B[m]_{jk}^{0}\right) \text{ for } i, j = 1, \ldots, k-1. \tag{2}$$

Row k is computes according to equation

$$B[m]_{kj}^{i+1} = \min\left(B[m]_{kj}^{i},\ B[m]_{ki}^{0} + B[m]_{ij}^{k-1}\right) \text{ for } i, j = 1, \ldots, k-1. \tag{3}$$

It should be noted that claim 1 does not apply to items (2) and (3). The second operation Ω_k calculates all elements of subblock $B[m]^{k-1}$ over row k and column k of computation level k, obtaining a block $B[m]^k$ of APSP in the subgraph on k vertices of cluster m. It uses the following formula to perform the calculation:

$$B[m]_{ij}^{k} = \min\left(B[m]_{ij}^{k-1},\ B[m]_{ik}^{k} + B[m]_{kj}^{k}\right) \text{ for } i, j = 1, \ldots, k-1.$$

We can describe the behaviour of *D*0*CG* as the sequence of pairs (Λ_k, Ω_k) of operations:

$$(\Lambda_1, \Omega_1), (\Lambda_2, \Omega_2), \ldots, (\Lambda_k, \Omega_k), \ldots, (\Lambda_S, \Omega_S). \tag{4}$$

Algorithm 5 displays the corresponding pseudocode. The loop along k represents sequence (4), in its body the first nest of loops along i and j is the operation Λ_k, and the second nest of loops over i and j is the operation Ω_k. The operations Λ_1 and Ω_1 do not change the block $B[m]^1$ compared to $B[m]^0$, so they are omitted, and we start with $k = 2$. The calculation levels of elements $B[m]_{i,j}$, $B[m]_{i,k}$ and $B[m]_{k,j}$ in the operation Ω_k of algorithm 5 satisfy claim 2, so the elements $B[m]_{i,j}$ are calculated correctly. The operation Λ_k correctly computes the elements $B[m]_{i,k}$ and $B[m]_{k,j}$ from level 0 to level k. We can conclude that algorithm 5 is correct.

Algorithm 5. Recurrent procedure *D*0*CG* for calculating diagonal block

$S \leftarrow C[m]$.size
for $k \leftarrow 2$ **to** S **do**
  for $i \leftarrow 1, \ldots, k-1$ **do**
    for $j \leftarrow 1, \ldots, k-1$ **do**
      $B[m]_{i,k}^{j+1} \leftarrow \min\left(B[m]_{i,k}^{j},\ B[m]_{i,j}^{k-1} + B[m]_{j,k}^{0}\right)$ // Λ_k
      $B[m]_{k,j}^{i+1} \leftarrow \min\left(B[m]_{k,j}^{i},\ B[m]_{k,i}^{0} + B[m]_{i,j}^{k-1}\right)$ // Λ_k
  for $i \leftarrow 1, \ldots, k-1$ **do**
    for $j \leftarrow 1, \ldots, k-1$ **do**
      $B[m]_{i,j}^{k} \leftarrow \min\left(B[m]_{i,j}^{k-1},\ B[m]_{i,k}^{k} + B[m]_{k,j}^{k}\right)$ // Ω_k
return $B[m]^S$.

Two nests of loops along i and j cannot be combined into a single nest of loops due to data dependencies: element $B[m]_{i,j}$ cannot be modified while it is used to modify all elements $B[m]_{i,k}$ and $B[m]_{k,j}$. To overcome this obstacle, we resynchronise (fig. 3, *b*) sequence (4) with the following sequence of pairs $(\Omega_{k-1}, \Lambda_k)$:

$$\Lambda_1, (\Omega_1, \Lambda_2), (\Omega_2, \Lambda_3), \ldots, (\Omega_{k-1}, \Lambda_k), \ldots, (\Omega_{S-1}, \Lambda_S), \Omega_S.$$

Now we can rewrite algorithm 5 to algorithm 6. The loop along k includes two nests of loops along i and j, which have the same iteration schemes. The first nest performs the operation Ω_{k-1}, and recalculates all elements of the subblock $B[m]^{k-1}$. The second nest performs the operation Λ_k and calculates column k and row k in the block $B[m]^k$. The two nests of loops can be merged, since $B[m]_{i,j}^{k-1}$ will not change due to the calculation of $B[m]_{i,k}^{k}$ and $B[m]_{k,j}^{k}$, and vice versa. As a result, we have obtained *D*0*CG*. The theorem is proved.

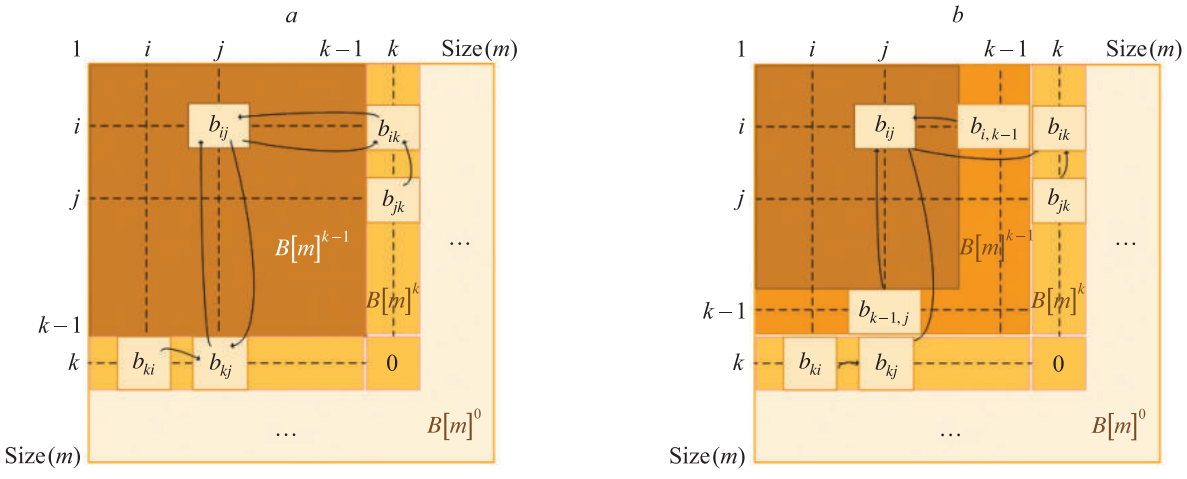

Fig. 3. Iterating the diagonal block $B[m]$ along calculation level k in *D0CG*:
a – adding vertex k to cluster m; *b* – resynchronised process of adding vertices

Competitive *BCal* and *D0CG* have different iteration schemes. The total number of iterations of the innermost loop of *BCal* is S^3. The total number of iterations of the innermost loop of *D0CG* is $\frac{6}{2+3\left(\text{size}(m)\right)^{-1}+\left(\text{size}(m)^2\right)^{-1}}$ times smaller than that of *BCal*. It tends to be 3, when $\text{size}(m)\to\infty$. *BCal* has $\text{size}(m)^2$ element accesses at each iteration of the loop along k. *D0CG* has k^2 accesses for $k=1,\ldots,size(m)$. Moreover, the element $B[m]_{i,j}$ is common to three assignments. In terms of data reference locality, *D0CG* outperforms *BCal* by up to 3 times. We further transform *D0CG* to incorporate vectorisation and parallelisation mechanisms.

Algorithm 6. Resynchronised *D0CG*

$S \leftarrow C[m]$.size
for $k \leftarrow 2$ **to** S **do**
  for $i \leftarrow 1, \ldots, k-1$ **do**
    for $j \leftarrow 1, \ldots, k-1$ **do**
      $B[m]_{i,j}^{k-1} \leftarrow \min\left(B[m]_{i,j}^{k-2},\ B[m]_{i,k-1}^{k-1} + B[m]_{k-1,j}^{k-1}\right)$ // Ω_{k-1}
  for $i \leftarrow 1, \ldots, k-1$ **do**
    for $j \leftarrow 1, \ldots, k-1$ **do**
      $B[m]_{i,k}^{j+1} \leftarrow \min\left(B[m]_{i,k}^{j},\ B[m]_{i,j}^{k-1} + B[m]_{j,k}^{0}\right)$ // Λ_k
      $B[m]_{k,j}^{i+1} \leftarrow \min\left(B[m]_{k,j}^{i},\ B[m]_{k,i}^{0} + B[m]_{i,j}^{k-1}\right)$ // Λ_k
for $i \leftarrow 1, \ldots, S-1$ **do**
  for $j \leftarrow 1, \ldots, S-1$ **do**
    $B[m]_{i,j}^{S} \leftarrow \min\left(B[m]_{i,j}^{S-1},\ B[m]_{i,S}^{S} + B[m]_{S,j}^{S}\right)$ // Ω_S
return $B[m]$.

New algorithm for calculating vertical cross-blocks

*C*1*CG* computes APSP from the nodes of cluster c to the nodes of cluster m and changes the vertical cross-block $B1 = B[c, m]$ of dimension $\text{size}(c)\times\text{size}(m)$ through diagonal block $B3 = B[m]$ of dimension $\text{size}(m)\times\text{size}(m)$. In this paper we assume that the block sizes are not equal and use the bridge-vertices of cluster m to speed up the computation of APSP. *C*1*CG* is a generalisation of *BCal* for vertical cross-blocks describing APSP in clustered directed graphs. It considers two clusters c and m and has two entries: a vertical crossing block $B[c, m]$ and a diagonal block $B[m]$. It returns the modified block $B[c, m]$.

Theorem 2. *Upon termination C*1*CG correctly computes the vertical cross-block* $B[c, m]$ *over the diagonal block* $B[m]$*, which describes APSP distances from the vertices of cluster c to the vertices of cluster m that pass through the input bridge-vertices of m.*

P r o o f. The blocks $B[c, m]$ and $B[m]$ have different dimensions: $\text{size}(c) \times \text{size}(m)$ and $\text{size}(m) \times \text{size}(m)$, respectively. The differences do not affect the fundamentals [12] of the shortest path calculation method (*BCal*) and the execution of min-plus operations on matrices. Therefore, *C*1*CG* is valid with respect to unequal block sizes.

The algorithm recalculates only one of the two blocks, i. e. $B[c, m]$ and thus relaxes the requirements of claim 1 to the ordering of the calculation levels of its elements. Therefore, it satisfies claim 1 in advance and is correct from this point of view. As a result, three loops along *k*, *i* and *j* can be reordered arbitrarily.

Algorithm 7. *C*1*CG* (calculation of vertical cross-block upon bridges and unequal block sizes)
for $i \leftarrow 1$ **to** $C[c].\text{size}$ **do**
  for $v \in C[m].\text{bridge.inall}$ **and** $k = C[m].\text{vert}[v].\text{index}$ **do**
    for $j \leftarrow 1$ **to** $C[m].\text{size}$ **do**
      $B[c, m]_{i,j} \leftarrow \min\left(B[c, m]_{i,j},\ B[c, m]_{i,k} + B[m]_{k,j}\right)$
return $B[c, m]$.

Now we prove that any shortest path from vertex *i* of cluster *c* to vertex *j* of cluster *m* passes through an input bridge-vertex *k* of cluster *m*. We assume that APSP between the inner vertices of cluster *m* have been already calculated. For the shortest path between *i* and *j* 2 cases are possible (fig. 4).

1. The path $p = (i, \ldots, k, \ldots, j)$ passes through vertices of the cluster *c*, then passes through the input bridge-vertex *k* and other vertices of cluster *m*, and finally reaches vertex *j*.

2. The path $p = (i, \ldots, w, \ldots, k, \ldots, j)$ passes through vertices of cluster *c*, then passes through vertices *w* of cluster *x* and possibly through other vertices of other clusters, then through an input bridge-vertex *k* of cluster *m*, and finally reaches the vertex *j*.

In both cases any shortest path between *i* and *j* passes through one of the input bridge-vertices of cluster *m*. That is why for loop along *k* in *C*1*CG* it is sufficient to traverse only the input bridge-vertices of cluster *m*. The theorem is proved.

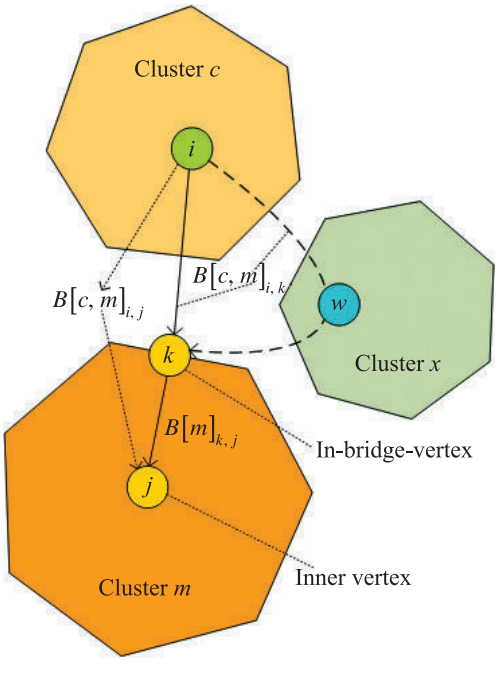

Fig. 4. Calculation of vertical cross-block through diagonal block (illustration of proof of theorem 2)

Corollary 1. *If cluster m has no input bridge-vertices, C*1*CG does not change the vertical cross-blocks* $B[c, m]$ *in column m of matrix B and hence is not applied to the column.*

*C*1*CG* gives a speedup in the computation of the vertical cross-blocks compared to the homogeneous blocked algorithm [14] depending on the share of the input bridge-vertices in the total number of bridge-vertices in cluster *m*.

New algorithm for calculating horizontal cross-blocks

*C*2*CG* computes APSP connecting vertices of cluster *m* with vertices of cluster *c*. It modifies a horizontal rectangular cross-block $B[m, c]$ of dimension $\text{size}(m) \times \text{size}(c)$ through the diagonal square block $B[m]$ of dimension $\text{size}(m) \times \text{size}(m)$. *C*2*CG* is a generalisation of *BCal* for horizontal cross-blocks describing shortest path segments in clustered directed graphs. It uses the output bridge-vertices of cluster *m* to speed up the shortest path computation.

Algorithm 8. *C*2*CG* (calculation of horizontal cross-block upon bridges and unequal block sizes)
for $i \leftarrow 1$ **to** $C[m]$.size **do**
  for $v \in C[m]$.bridge.outall **and** $k = C[m]$.vert$[v]$.index **do**
    for $j \leftarrow 1$ **to** $C[c]$.size **do**
      $B[m, c]_{i,j} \leftarrow \min\left(B[m, c]_{i,j},\ B[m]_{i,k} + B[m, c]_{k,j}\right)$
return $B[m, c]$.

Theorem 3. *Upon termination C*2*CG correctly computes horizontal cross-block* $B[m, c]$ *through diagonal block* $B[m]$*, which describes APSP distances from vertices of cluster m to vertices of cluster c that pass through the output bridge-vertices of m.*

P r o o f. Although the blocks $B[m, c]$ and $B[m]$ have different sizes [12], these differences do not affect the correctness of the shortest path calculation method and min-plus operations on matrices as it is done for *BCal*. Therefore, *C*2*CG* is correct from this point of view.

The algorithm relaxes the requirements of claim 1 for the order of the matrix element calculation levels, since it recalculates only one block $B[c, m]$ and does not change other block $B[m]$ at the same time. Therefore, it correctly computes $B[c, m]$ for any order of three loops along variables *i*, *k* and *j*.

Now we suppose that APSP between the inner vertices of cluster *c* and between the inner vertices of cluster *m* have been already calculated. At this assumption, we prove that any shortest path from vertex *i* of cluster *m* to vertex *j* of cluster *c* passes through an output bridge-vertex *k* of cluster *m*. For the shortest path between *i* and *j* 2 cases are possible (fig. 5).

1. The path $p = (i, \ldots, k, \ldots, j)$ passes through vertices of cluster *m* including an output bridge-vertex *k*, then passes through vertices of cluster *c*, and finally reaches vertex *j* of the cluster.

2. The path $p = (i, \ldots, k, \ldots, w, \ldots, j)$ passes through vertices of cluster *m* including the output bridge-vertex *k*, then passes through one or more vertices *w* of cluster *x* and may be other vertices of other clusters, then passes through vertices of cluster *c*, and finally reaches vertex *j* of the cluster.

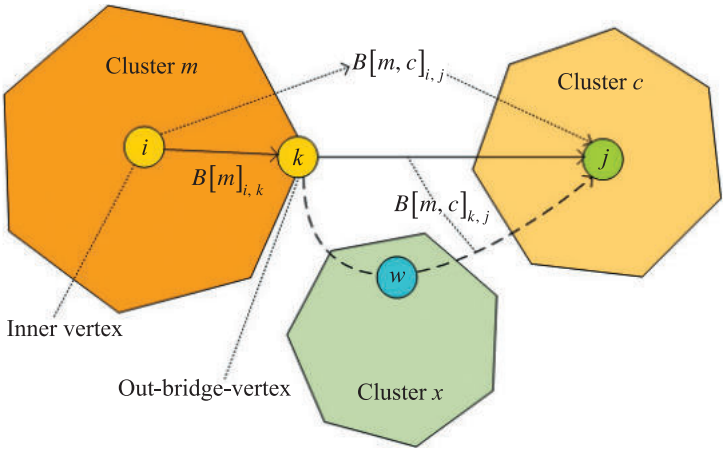

Fig. 5. Calculation of horizontal cross-block through diagonal block
(illustration of proof of theorem 3)

In both cases any shortest path between i and j passes through one of the output bridge-vertices of cluster m. This is why for the iteration scheme of the loop along k in *C*2*CG* it is sufficient to traverse only the output bridge-vertices of cluster m. The theorem is proved.

Corollary 2. *If a cluster m has no output bridge-vertices, C*2*CG does not change the horizontal cross-blocks* $B[m, c]$ *in row m of matrix B and hence is not applied to the row.*

*C*2*CG* provides the speedup in computation of the horizontal cross-blocks in comparison to the homogeneous blocked algorithm [14] depending on the share of the output bridge-vertices in the total number of bridge-vertices in cluster m.

New algorithm for calculating peripheral blocks

*P*3*CG* computes shortest-path segments. The segments connect vertices of cluster c with vertices of cluster e, passing through vertices of cluster m. The algorithm modifies a rectangular block $B[c, e]$ of dimension $\text{size}(c) \times \text{size}(e)$ using two rectangular blocks $B[c, m]$ and $B[m, e]$ of the dimension $C[c].\text{size} \times C[m].\text{size}$ and $C[m].\text{size} \times C[e].\text{size}$, respectively. *P*3*CG* is a generalisation of *FW* for clustered directed graphs. It uses the input or output bridge-vertices of cluster m to speed up the computations.

The algorithm consists of three nested loops along variables i, k and j and includes one assignment statement. The loops along i and j traverse all vertices of clusters c and e, respectively. The loop along k traverses the vertices of cluster m that belong to the subset $C[m].\text{bridge.best}$. The subset is one of two vertex subsets that has the minimum size: $C[m].\text{bridge.best} = C[m].\text{bridge.inall}$ if $|C[m].\text{bridge.inall}| = |C[m].\text{bridge.outall}|$, and $C[m].\text{bridge.best} = C[c].\text{bridge.outall}$, otherwise.

Algorithm 9. *P*3*CG* (calculation of peripheral rectangular block upon bridges and unequal block sizes)
for $i \leftarrow 1$ **to** $C[c].\text{size}$ **do**
  for $v \in C[m].\text{bridge.best}$ **and** $k \leftarrow C[m].\text{vert}[v].\text{index}$ **do**
    for $j \leftarrow 1$ **to** $C[e].\text{size}$ **do**
      $B[c, e]_{i,j} \leftarrow \min\left(B[c, e]_{i,j},\ B[c, m]_{i,k} + B[m, e]_{k,j}\right)$
return $B[c, e]$.

Theorem 4. *Upon termination P*3*CG correctly computes block* $B[c, e]$ *over blocks* $B[c, m]$ *and* $B[m, e]$*, which describe the shortest path segments from the vertices of cluster c to the vertices of cluster e that pass through the best bridge-vertices of cluster m.*

P r o o f. Although, unlike *BCal*, the input blocks $B[c, e]$, $B[c, m]$ and $B[m, e]$ have different sizes in *P*3*CG*, the differences do not affect the shortest path calculation technique [12] and min-plus operations on matrices as is done in *BCal*. Therefore, *P*3*CG* is correct regarding different block sizes, since *BCal* is proven to be correct.

The algorithm relaxes the requirements of claim 1 for the order of the calculation levels of the distance matrix elements, since it recalculates only the elements of one block $B[c, e]$ and does not simultaneously change the elements of the other two blocks $B[c, m]$ and $B[m, e]$. Therefore, it correctly computes the block $B[c, e]$ with respect to the calculation levels for any reordering of three loops along variables i, k and j.

Now we prove that any shortest path $p = (i, \ldots, k, \ldots, j)$, where $i \in C[c].\text{vert}$, $k \in C[m].\text{vert}$ and $j \in C[e].\text{vert}$, includes the input bridge-vertex of cluster m. As shown in fig. 6, there are 4 cases of the shortest path passing through three clusters.

1. The strait path through clusters c, m and e is $p = (i, \ldots, k, \ldots, j)$. This means that the path starting from i passes through the vertices of cluster c, enters cluster m through the input bridge-vertex k, exits cluster m through one of the output bridge-vertices, and finally goes to vertex j of cluster e.

2. The path is $p = (i, \ldots, w, \ldots, k, \ldots, j)$, where $w \in C[x].\text{vert}$, and x is a cluster other than c, m and e. The path goes from i through vertices of cluster c, goes through vertices w of cluster x (might be several clusters), enters cluster m through vertex k, exits it through an output bridge-vertex, and finally passes through vertices of cluster e to vertex j.

3. The path is $p = (i, \ldots, k, \ldots, z, \ldots, j)$, where $z \in C[y].\text{vert}$, and y is a cluster other than c, m and e. The path goes from i through vertices of cluster c, enters cluster m through vertex k, exits it through an output bridge-vertex, passes through vertices z of cluster y (there may be several clusters), and finally goes through vertices of cluster e to vertex j.

4. The path is $p = (i, \ldots, w, \ldots, k, \ldots, z, \ldots, j)$, where w and z are vertices of clusters other than c, m and e. The path goes from i through vertices of cluster c, passes through vertices w of cluster x (there may be several

clusters), enters cluster m through vertex k, exits it through an output bridge-vertex, passes through vertices z of cluster y (there may be several clusters), and finally passes through vertices of cluster e to vertex j.

Having considered these 4 cases, we can conclude that no path from i to j that does not pass through an input bridge-vertex of cluster m. Therefore, all iterations of loop along k in *P*3*CG* may correspond to the input bridge-vertices of cluster m.

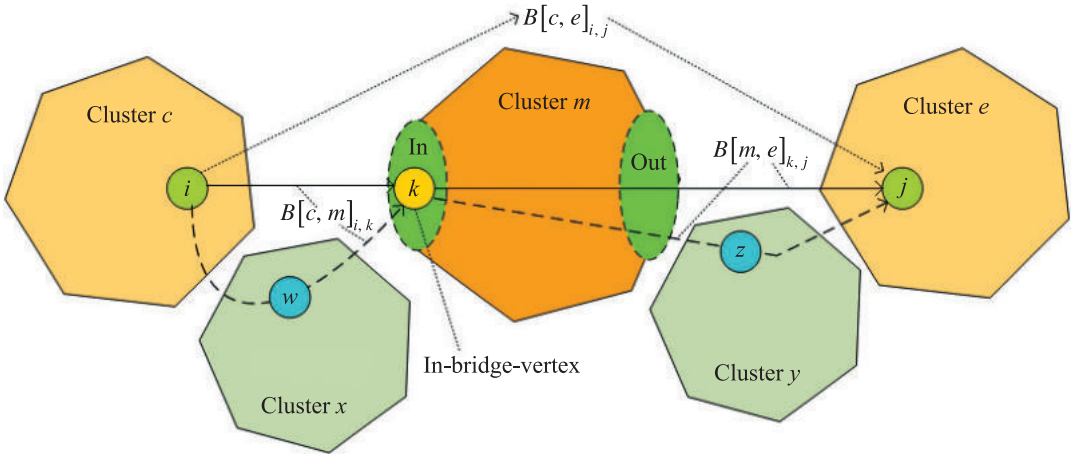

Fig. 6. Calculation of peripheral block over two cross-blocks through input bridge-vertices of cluster m (illustration of proof of theorem 4)

Similarly (fig. 7), it can be proved that any shortest path $p = (i, \ldots, k, \ldots, j)$, where $i \in C[c].\text{vert}$, $k \in C[m].\text{vert}$ and $j \in C[e].\text{vert}$, includes an output bridge-vertex of cluster m. This means that the iterations of loop along k in *P*3*CG* can correspond to the output bridge-vertices of cluster m. We have two alternatives for the loop iteration scheme. To speed up the computations, we choose the subset $C[m].\text{bridge.best}$ that has the smallest size. The theorem is proven.

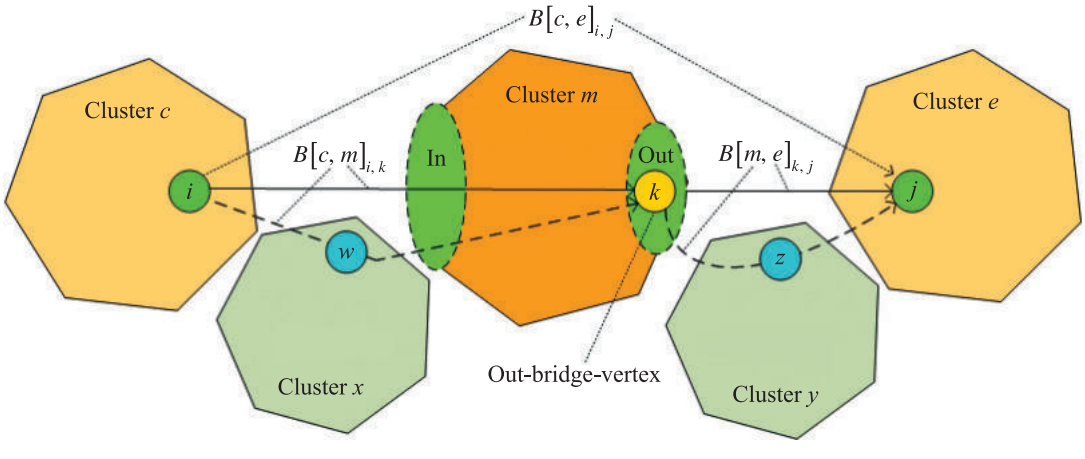

Fig. 7. Calculation of peripheral block over two cross-blocks through output bridge-vertices of cluster m (illustration of proof of theorem 4)

Corollary 3. *If cluster m has no input or output bridge-vertices, P*3*CG does not change the peripheral blocks* $B[c, e]$, $c, e \in \{1, \ldots, M\}$, $c \neq m$ *and* $e \neq m$, *of matrix B and therefore is not applied to all the peripheral blocks.*

The smaller the number of bridge-vertices in $C[m].\text{bridge.best}$ than the less central processing unit time the *P*3*CG* consumes.

Experimental results and comparison of algorithms and their implementations

We developed in C++ language and used two versions (single-threaded and multi-threaded OpenMP (version 4.5)) of implementations of the proposed *HBSPCG* and, for comparison, implementations of previously known algorithms. The source code was compiled by GNU Compiler Collection (version 14.2.0) with auto-vectorisation

enabled. The paper presents the results of experiments performed on two computers: MacBook M1 Max and server consisting of two Intel Xeon E5-2620v4 processors (each has 8 cores and 16 physical threads).

The article describes experiments conducted on four random directed simple weighted graphs decomposed into dense weakly connected clusters of different sizes (table 1). Judging by the number of edges in the graphs and the number of edges between clusters, all clusters are dense subgraphs. The edge density in all four graphs was in the range of 0.003 55–0.012 51, meaning that all graphs were sparse.

Table 1

A sample of four random sparse graphs consisting of tens of dense clusters

Number of graph	Vertices	Clusters	Edges	Density	Bridge-vertices	Bridge-edges
1	4800	20	288 245	0.012 51	567	621
2	4800	41	153 858	0.006 68	620	687
3	9600	40	644 198	0.006 99	3452	2374
4	9600	80	326 779	0.003 55	3550	2505

Table 2 shows results obtained on MacBook M1 Max. In case of single-threaded implementations the speedup of the proposed heterogeneous blocked algorithm *HBSPCG* (clustered graph, bridge-vertices) compared to the known homogeneous blocked algorithm *BSPCG* [14] (clustered graph, bridge-vertices) is of 1.62–1.94 times. In case multi-threaded OpenMP implementations the speedup is 1.87–1.97 times.

Table 2

Runtimes of algorithms *BSPCG* and *HBSPCG* on MacBook M1 Max

Number of graph	Single-threaded implementations			Multi-threaded OpenMP implementations		
	BSPCG, s	*HBSPCG*, s	Speedup of *HBSPCG* over *BSPCG*, times	*BSPCG*, s	*HBSPCG*, s	Speedup of *HBSPCG* over *BSPCG*, times
1	3.21	1.65	1.94	0.69	0.35	1.97
2	2.70	1.66	1.62	0.48	0.26	1.88
3	37.30	19.52	1.91	5.54	2.87	1.93
4	34.84	21.15	1.65	5.31	2.84	1.87

The results obtained on the server are shown in the table 3. *HBSPCG* outperformed *BSPCG* by 1.58–1.66 times for single-threaded implementations and by 1.29–1.64 times for multi-threaded OpenMP implementations. Table 3 also provides a comparison of *HBSPCG* with the classical *BFW* with equal block sizes. The speedup of *HBSPCG* over *BFW* was in the range of 4.17–8.18 for single-threaded implementations and 3.91–6.36 for multi-threaded OpenMP implementations.

Table 3

Comparison of *BSPCG*, *HBSPCG* and *BFW* on server with two Intel Xeon E5-2620v4 processors

Number of graph	Single-threaded implementations				Multi-threaded OpenMP implementations			
	BSPCG, s	*HBSPCG*, s	Speedup of *HBSPCG* over *BSPCG*, times	Speedup of *HBSPCG* over *BFW*, times	*BSPCG*, s	*HBSPCG*, s	Speedup of *HBSPCG* over *BSPCG*, times	Speedup of *HBSPCG* over *BFW*, times
1	11.07	6.28	1.76	8.18	0.91	0.57	1.59	6.36
2	11.18	6.82	1.64	7.24	1.05	0.82	1.29	4.13
3	148.08	89.06	1.66	4.59	9.62	5.86	1.64	4.52
4	149.95	94.90	1.58	4.17	10.50	6.48	1.62	3.91

Conclusions

The *FW* family of algorithms, which solve the problem of APSP has cubic time complexity and quadratic memory complexity regardless of the number of edges in the graph that creates obstacles for processing real large graphs on multi-processor systems. The goal of the *BFW* is to provide parallelism and efficient use of

the hierarchical processor memory. It is most efficient on dense graphs and has losses on sparse graphs. Many works and publications are devoted to achievements in the field of reducing the computational resources consumed by the blocked algorithm. In the paper we propose a heterogeneous version of such an algorithm that considers the features of large clustered directed weighted graphs, which are divided into dense clusters of different sizes, weakly connected by bridge-vertices and bridge-edges. The algorithm distinguishes four types of blocks and exploits their unique features in a separate procedure for each block to speed up the computation of APSP and improve the locality of references to data. This allowed us to reduce the runtime by approximately twice on the MacBook M1 Max computer compared to the well-known homogeneous *BFW* on the clustered graphs and to reduce the runtime up to eight times on the server compared to the classical *BFW* with equal block sizes.

References

1. Madkour A, Aref WG, Rehman F, Rahman MA, Basalamah SM. Survey of shortest-path algorithms. arXiv:1705.02044vl [Preprint]. 2017 [cited 2025 May 2]: [26 p.]. Available from: https://arxiv.org/abs/1705.02044.

2. Prihozhy AA. Synthesis of quantum circuits based on incompletely specified functions and *if*-decision diagrams. *Journal of the Belarusian State University. Mathematics and Informatics.* 2021;3:84–97. DOI: 10.33581/2520-6508-2021-3-84-97.

3. Floyd RW. Algorithm 97: shortest path. *Communications of the ACM*. 1962;5(6):345. DOI: 10.1145/367766.368168.

4. Warshall S. A theorem on boolean matrices. *Journal of the ACM*. 1962;9(1):11–12. DOI: 10.1145/321105.321107.

5. Venkataraman GA, Sahni S, Mukhopadhyaya S. A blocked all-pairs shortest-paths algorithm. *Journal of Experimental Algorithmics*. 2003;8:857–874. DOI: 10.1145/996546.996553.

6. Park J-S, Penner M, Prasanna VK. Optimizing graph algorithms for improved cache performance. *IEEE Transactions on Parallel and Distributed Systems*. 2004;15(9):769–782. DOI: 10.1109/TPDS.2004.44.

7. Лиходед НА, Сипейко ДС. Обобщенный блочный алгоритм Флойда – Уоршелла. *Журнал Белорусского государственного университета. Математика. Информатика.* 2019;3:84–92. DOI: 10.33581/2520-6508-2019-3-84-92.

8. Djidjev H, Chapuis G, Andonov R, Thulasidasan S, Lavenier D. All-pairs shortest path algorithms for planar graph for GPU-accelerated clusters. *Journal of Parallel and Distributed Computing*. 2015;85:91–103. DOI: 10.1016/j.jpdc.2015.06.008ff.

9. Yang S, Liu X, Wang Y, He X, Tan G. Fast all-pairs shortest paths algorithm in large sparse graph. In: Association for Computing Machinery. *ICS'23. Proceedings of the 37th International conference on supercomputing; 2023 June 21–23; Orlando, USA.* New York: Association for Computing Machinery; 2023. p. 277–288. DOI: 10.1145/3577193.3593728.

10. Прихожий АА, Карасик ОН. Разнородный блочный алгоритм поиска кратчайших путей между всеми парами вершин графа. *Системный анализ и прикладная информатика.* 2017;3:68–75. EDN: ZWMSYN.

11. Prihozhy AA, Karasik ON. Advanced heterogeneous block-parallel all-pairs shortest path algorithm. *Proceedings of BSTU. Issue 3, Physics and Mathematics. Informatics.* 2023;1:77–83. DOI: 10.52065/2520-6141-2023-266-1-13.

12. Prihozhy AA, Karasik ON. New blocked all-pairs shortest paths algorithms operating on blocks of unequal sizes. *System Analysis and Applied Information Science.* 2023;(4):4–13. DOI: 10.21122/2309-4923-2023-4-4-13.

13. Prihozhy AA, Karasik ON. Blocked algorithm of shortest paths search in sparse graphs partitioned into unequally sized clusters. In: Bogush VA, Dik SK, Likhachevskii DV, Kazak TV, Piskun GA, editors. *Big data and high-level analysis. Collection of scientific articles of the 10th International scientific and practical conference; 2024 March 13; Minsk, Belarus.* Minsk: Belarusian State University of Informatics and Radioelectronics; 2024. p. 262–271. EDN: FSMHWS.

14. Karasik ON, Prihozhy AA. Blocked algorithm of finding all-pairs shortest paths in graphs divided into weakly connected clusters. *System Analysis and Applied Information Science.* 2024;2:4–10. DOI: 10.21122/2309-4923-2024-2-4-10.

15. Carlson T, Wong G. Optimization of the Floyd – Warshall shortest path algorithm. In: Arabnia HR, Deligiannidis L, Amirian S, Ghareh Mohammadi F, Shenavarmasouleh F, editors. *Foundations of computer science and frontiers in education: computer science and computer engineering. Proceedings of the 20th International conference on foundations of computer science and 20th International conference on frontiers in education; 2024 July 22–25; Las Vegas, USA.* Las Vegas: Springer; 2025. p. 84–90 (Communications in computer and information science; volume 2261).

16. Sangeetha DP, Sekar S, Parvathy PR, GaneshBabu SRTR, Muthulekshmi M. Optimizing shortest paths in big data using the Floyd – Warshall algorithm. In: GL BAJAJ Group of Institutions. *Proceedings of the International conference on intelligent control, computing and communications; 2025 February 13–14; Mathura, India.* [S. l.]: IEEE; 2025. p. 382–387. DOI: 10.1109/IC363308.2025.10957179.

17. Liu G. Solving the all pairs shortest path problem after minor update of a large dense graph. arXiv:2412.15122v6 [Preprint]. 2025 [cited 2025 November 16]: [10 p.]. Available from: https://arxiv.org/pdf/2412.15122.

18. Kumar S, Karthik S, Srilakshmi S, Dharun Viginesh P. Performance analysis of Floyd – Warshall algorithm: sequential and parallel execution using intel oneAPI. In: RVS Technical Campus. *Proceedings of the 8th International conference on electronics, communication and aerospace technology; 2024 August 6; Coimbatore, India.* [S. l.]: IEEE; 2024. p. 205–211. DOI: 10.1109/ICECA63461.2024.10800787.

19. Карасик ОН, Прихожий АА. Потоковый блочно-параллельный алгоритм поиска кратчайших путей на графе. *Доклады Белорусского государственного университета информатики и радиоэлектроники.* 2018;2:77–84. EDN: YVOTCR.

20. Prihozhy AA. Generation of shortest path search dataflow networks of actors for parallel multi-core implementation. *Informatics*. 2023;20(2):65–84. DOI: 10.37661/1816-0301-2023-20-2-65-84.

21. Karasik ON, Prihozhy AA. Tuning block-parallel all-pairs shortest path algorithm for efficient multi-core implementation. *System Analysis and Applied Information Science.* 2022;3:57–65. DOI: 10.21122/2309-4923-2022-3-57-65.

22. Prihozhy AA, Karasik ON. Influence of shortest path algorithms on energy consumption of multi-core processors. *System Analysis and Applied Information Science.* 2023;2:4–12. DOI: 10.21122/2309-4923-2023-2-4-12.

Received 10.05.2025 / revised 25.08.2025 / accepted 19.11.2025.