Cent. Eur. J. Comp. Sci. • 3(4) • 2013 • 149-157
DOI: 10.2478/s13537-013-0110-4

VERSITA

Central European Journal of **Computer Science**

An all-pairs shortest path algorithm for bipartite graphs

Research Article

Svetlana Torgasin*, Karl-Heinz Zimmermann†

Hamburg University of Technology,
21071 Hamburg, Germany

Received 15 March 2013; accepted 04 November 2013

Abstract: Bipartite graphs are widely used for modeling of complex structures in biology, engineering, and computer science. The search for shortest paths in such structures is a highly demanded procedure that requires optimization. This paper presents a variant of the all-pairs shortest path algorithm for bipartite graphs. The method is based on the distance matrix product and improves the general algorithm by exploiting the graph topology. The space complexity is reduced by a factor of at least four and the time complexity decreased by almost an order of magnitude when compared with the basic APSP algorithm.

Keywords: bipartite graph • tropical (min-plus) algebra • shortest path • distance matrix product

1. Introduction

The all-pairs shortest path (APSP) problem is one of the fundamental algorithmic problems in graph theory. The prominent approaches for solving this problem are the algorithms of Floyd-Warshall [1, 2] and Johnson [3]. The computational complexity of the first method is $O(|V|^3)$ and of the second one $O(|V|^2 \log|V|+|V||E|)$. One main strain of the recent methods in this field is to achieve a sub-cubic complexity. Several of the proposed solution methods are based on specific data structures and properties of computer architecture such as table lookups [4], bit-level parallelism [5] or parallel computations [6, 7]. Other methods follow a heuristic approach, for instance, by solving the all-pairs almost shortest path problem, which is a relaxation of the original problem allowing certain deviations from the optimal cost [8]. A detailed survey of the methods that were developed so far can be found in Dragan [9] and Zwick [10].

With the progress in application areas like networking or logistics, the need to solve the APSP problem for specific classes of graphs has increased. These methods exploit the structure of the graph topology exhibited by certain types of graphs such as chordal, permutation or distance-hereditary graphs. Moreover, restrictions on edge costs, such as unit cost (unweighted graph) or bounded positive values, are widely used to improve time complexity. Solutions for specific types of bipartite graphs have been proposed by Chin-Wen and Chang [11], and Chen [6]. The first work has presented a solution for unweighted chordal bipartite graphs, and the second one has proposed sequential and parallel algorithms for unweighted bipartite permutation graphs.

This paper provides a novel method to solving the APSP problem for real-weighted bipartite graphs without negative cycles.

* *E-mail: torgasin@tuhh.de (Corresponding author)*
† *E-mail: k.zimmermann@tuhh.de*

2. The distance matrix of a graph

The graphs considered in the APSP problem can be directed or undirected. An undirected graph is an ordered pair $G = (V, E)$ consisting of a set V of vertices and a set E of edges. An edge $\{u, v\}$ in G is a 2-element subset of V relating two vertices and is given by an unordered pair of vertices denoted as a (commutative) word $uv (= vu)$. A directed graph is an ordered pair $G = (V, E)$ comprising a set V of vertices and a set E of edges defined as a subset of $V \times V$. An edge (u, v) in G connects two vertices and is specified as an ordered pair that is depicted as a word uv. A weighted graph is a triple $G = (V, E, d)$, where (V, E) is a (directed or undirected) graph and $d : E \longrightarrow \mathbb{R} \cup \{\infty\}$ is a mapping that associates a cost to each edge given as a real number or ∞ (infinity).
Let $V = \{v_1, \ldots, v_n\}$. The cost mapping d can be represented by the $n \times n$ cost matrix $D = (d_{ij})$, where

$$d_{ij} = \begin{cases} d(v_i v_j), & \text{if } v_i v_j \in E, \\ 0, & \text{if } i = j, \\ \infty, & \text{otherwise.} \end{cases} \tag{1}$$

A path in G is a sequence of vertices $v_{i_0}, v_{i_1}, \ldots, v_{i_k}$ such that consecutive vertices form an edge $v_{i_j} v_{i_{j+1}}$ in G, where $0 \leqslant j \leqslant k-1$. The length of a path in G is the number of edges in the path. On the other hand, the cost of a path in G is the sum of costs of the edges in the path. The distance from a vertex v_i to a vertex v_j in G is the minimum cost over all paths from v_i to v_j in G denoted by d^*_{ij}. The all-pairs shortest path problem is to find the paths of lowest cost for each pair of vertices in G. The result is often represented by the distance matrix $D^* = (d^*_{ij})$ that contains the distances for each pair of vertices as entries.
In the following, it is assumed that the graph under consideration has no negative cycles. A negative cycle is a cycle whose edge sum is negative. If the shortest path between two vertices contains a negative cycle, its length will not be well-defined. Nevertheless, if negative cycles exist, the Floyd-Warshall algorithm and also the subsequent algorithm can be used to detect them.
The approach to be presented is a distance product APSP method known since the late 1950's [12] and widely considered in literature [7, 9, 10]. This method is based on the distance matrix multiplication. The distance matrix product of two square $n \times n$ matrices $A = (a_{ij})$ and $B = (b_{ij})$ is an $n \times n$ matrix $C = (c_{ij}) = A \times B$ with

$$c_{ij} = \min\{a_{ik} + b_{kj} \mid 1 \leqslant k \leqslant n\}, \quad 1 \leqslant i, j \leqslant n. \tag{2}$$

The distance matrix D^* for a graph can be computed by repeated distance matrix multiplications of the cost matrix D:

$$D^* = D^{\times n-1}. \tag{3}$$

The exponent $n-1$ in Equation (3) is the maximal length of a non-cyclic path in a graph with n vertices. Longer paths need not be considered, since their cost can only be larger unless the graph contains negative cycles.
The repeated squaring technique allows to find the n-th power of a matrix in $\log_2(n)$ steps [13]. Then, the time complexity for evaluating the distance matrix becomes $O(\ell(n) \log_2 n)$, where $\ell(n)$ is the time complexity of the multiplication of $n \times n$ matrices. Note that for a graph with negative cycles, the distance matrix products would contain negative entries on the main diagonal, and those entries could be detected during the multiplication.
The distance products can also be calculated in the tropical algebra $(\mathbb{R} \cup \{\infty\}, \oplus, \odot)$, which was first studied by Imre Simon [14]. As a set this is just the set $\mathbb{R}$ of real numbers together with an extra element ∞ (infinity). However, the usual arithmetic operations are replaced by the tropical operations

$$x \oplus y = \min\{x, y\} \quad \text{and} \quad x \odot y = x + y. \tag{4}$$

The tropical algebra is a semiring with ∞ and 0 as the neutral elements of addition and multiplication, respectively, since for each $x \in \mathbb{R} \cup \{\infty\}$,

$$x \oplus \infty = x \quad \text{and} \quad x \odot 0 = x. \tag{5}$$

The tropical algebra has recently received considerable attention in combinatorics, algebraic geometry and related fields [15]. The tropical operations can be carried forward to matrix calculations. Indeed, if $A = (a_{ij})$ and $B = (b_{ij})$ are tropical $n \times n$ matrices, the tropical sum $C = (c_{ij}) = A \oplus B$ and the tropical product $D = (d_{ij}) = A \odot B$ are respectively given by

$$c_{ij} = a_{ij} \oplus b_{ij} \quad \text{and} \quad d_{ij} = \bigoplus_k a_{ik} \odot b_{kj}, \tag{6}$$

which can be written in conventional terms as

$$c_{ij} = \min\{a_{ij}, b_{ij}\} \quad \text{and} \quad d_{ij} = \min\{a_{ik} + b_{kj} \mid 1 \leqslant k \leqslant n\}. \tag{7}$$

The tropical product is also defined for rectangular matrices with appropriate dimensions.
Note that the tropical identity matrix is

$$U = \begin{pmatrix} 0 & \infty & \cdots & \infty \\ \infty & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \infty \\ \infty & \cdots & \infty & 0 \end{pmatrix}, \tag{8}$$

since for each $m \times n$ matrix A the following holds: $A \odot U_n = A = U_m \odot A$, where U_n denotes the $n \times n$ matrix U.
Moreover, the tropical sum $A \oplus U$ touches only the positive entries on the main diagonal of the matrix A setting them to zero, since $x \oplus 0 = 0$ for each $x \geqslant 0$, $x \oplus 0 = x$ for each $x \leqslant 0$, and $x \oplus \infty = x$ for each x.
Note that the distance matrix product (2) equals the tropical multiplication of matrices (6) and thus the distance matrix in Equation (3) can be calculated in the tropical algebra as a tropical matrix product, i.e.,

$$D^* = D^{\odot n-1}. \tag{9}$$

The objective in the next section is to calculate the distance matrix for bipartite graphs.

3. The distance matrix of a bipartite graph

A bipartite graph is a graph $G = (V, E)$, where vertex set V can be partitioned into two non-empty subsets, V_1 and V_2 such that every edge connects a vertex in one set to a vertex in the other set (Fig. 1).

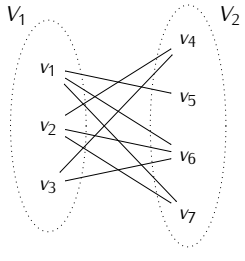


Figure 1. A bipartite graph.

Let $G = (V, E)$ be a bipartite graph with vertex set $V = V_1 \dot{\cup} V_2$. Put $n_1 = |V_1|$ and $n_2 = |V_2|$. The corresponding cost matrix D can be arranged so that the first n_1 rows and columns are labeled with the vertices in V_1, and the remaining ones with the vertices in V_2. In view of structure of bipartite graphs, the matrix D decomposes into four blocks as follows:

$$D = \begin{pmatrix} U_{n_1} & D_1 \\ D_2 & U_{n_2} \end{pmatrix}, \tag{10}$$

where D_1 is an $n_1 \times n_2$ matrix, D_2 is an $n_2 \times n_1$ matrix, and the two U-blocks are tropical identity matrices of the respective dimensions. If the graph is undirected, D_2 is the transposed matrix of D_1.
We may assume without restriction that $n_1 \leqslant n_2$. Then each non-cyclic path in G can have length at most $2n_1$, since each path in G is an alternating sequence of vertices in V_1 and V_2 and if the path is not a cycle, each vertex in V_1 will be traversed at most once. It follows that the $2n_1$-th tropical power of the cost matrix D provides a solution of the APSP problem:

$$D^* = D^{\odot 2n_1}. \tag{11}$$

The next result describes how the tropical powers of the distance matrix of even order can be computed.
In the following, we put

$$(D_1D_2)_U = D_1 \odot D_2 \oplus U_{n_1} \quad \text{and} \quad (D_2D_1)_U = D_2 \odot D_1 \oplus U_{n_2}. \tag{12}$$

Proposition 1.
Let $k \geqslant 1$ be an integer. The $2k$-th tropical power of the cost matrix D in (10) is given by

$$D^{\odot 2k} = \begin{pmatrix} L_1^{(2k)} & D_1^{(2k)} \\ D_2^{(2k)} & L_2^{(2k)} \end{pmatrix}, \tag{13}$$

where

$$\begin{aligned} L_1^{(2k)} &= (D_1D_2)_U^{\odot k}, \\ L_2^{(2k)} &= (D_2D_1)_U^{\odot k}, \\ D_1^{(2k)} &= (D_1D_2)_U^{\odot k-1} \odot D_1, \\ D_2^{(2k)} &= (D_2D_1)_U^{\odot k-1} \odot D_2. \end{aligned} \tag{14}$$

Proof. We use induction on k. For $k = 1$, the second power of D is given by

$$D^{\odot 2} = \begin{pmatrix} U_{n_1} & D_1 \\ D_2 & U_{n_2} \end{pmatrix} \odot \begin{pmatrix} U_{n_1} & D_1 \\ D_2 & U_{n_2} \end{pmatrix} = \begin{pmatrix} U_{n_1} \oplus D_1 \odot D_2 & D_1 \oplus D_1 \\ D_2 \oplus D_2 & D_2 \odot D_1 \oplus U_{n_2} \end{pmatrix}. \tag{15}$$

But the tropical matrix sum is idempotent and so by (12) we yield

$$L_1^{(2)} = (D_1D_2)_U,\ L_2^{(2)} = (D_2D_1)_U,\ D_1^{(2)} = D_1,\ \text{and}\ D_2^{(2)} = D_2. \tag{16}$$

Assume that the assertion holds for the tropical powers of D of at most $2k$. By using (16), the $2(k+1)$-th tropical power of D can be calculated as follows:

$$D^{\odot 2(k+1)} = \begin{pmatrix} L_1^{(2k)} & D_1^{(2k)} \\ D_2^{(2k)} & L_2^{(2k)} \end{pmatrix} \odot \begin{pmatrix} (D_1D_2)_U & D_1 \\ D_2 & (D_2D_1)_U \end{pmatrix}. \tag{17}$$

By induction, we obtain

$$\begin{aligned} L_1^{(2(k+1))} &= L_1^{(2k)} \odot (D_1 D_2)_U \oplus D_1^{(2k)} \odot D_2 \\ &= (D_1 D_2)_U^{\odot k} \odot (D_1 D_2)_U \oplus (D_1 D_2)_U^{\odot k-1} \odot D_1 \odot D_2 \\ &= (D_1 D_2)_U^{\odot k-1} \odot [(D_1 D_2)_U \odot (D_1 D_2)_U \oplus D_1 \odot D_2]^1 \\ &= (D_1 D_2)_U^{\odot k+1}, \end{aligned}$$

$$\begin{aligned} L_2^{(2(k+1))} &= D_2^{(2k)} \odot D_1 \oplus L_2^{(2k)} \odot (D_2 D_1)_U \\ &= (D_2 D_1)_U^{\odot k-1} \odot D_2 \odot D_1 \oplus (D_2 D_1)_U^{\odot k} \odot (D_2 D_1)_U \\ &= (D_2 D_1)_U^{\odot k-1} [D_2 \odot D_1 \oplus (D_2 D_1)_U \odot (D_2 D_1)_U] \\ &= (D_2 D_1)_U^{\odot k+1}, \end{aligned}$$

$$\begin{aligned} D_1^{(2(k+1))} &= L_1^{(2k)} \odot D_1 \oplus D_1^{(2k)} \odot (D_2 D_1)_U \\ &= (D_1 D_2)_U^{\odot k} \odot D_1 \oplus (D_1 D_2)_U^{\odot k-1} \odot D_1 \odot (D_2 D_1)_U \\ &= (D_1 D_2)_U^{\odot k-1} \odot [(D_1 D_2)_U \odot D_1 \oplus D_1 \odot (D_2 D_1)_U] \\ &= (D_1 D_2)_U^{\odot k} \odot D_1, \end{aligned}$$

and

$$\begin{aligned} D_2^{(2(k+1))} &= D_2^{(2k)} \odot (D_1 D_2)_U \oplus L_2^{(2k)} \odot D_2 \\ &= (D_2 D_1)_U^{\odot k-1} \odot D_2 \odot (D_1 D_2)_U \oplus (D_2 D_1)_U^{\odot k} \odot D_2 \\ &= (D_2 D_1)_U^{\odot k-1} \odot [D_2 \odot (D_1 D_2)_U \oplus (D_2 D_1)_U \odot D_2] \\ &= (D_2 D_1)_U^{\odot k} \odot D_2. \end{aligned}$$

□

The next assertion provides some expressions that will allow us to unify terms in the $2k$-th tropical power of the cost matrix D.

Proposition 2.
For any integer $k \geqslant 1$,

$$(D_2 D_1)_U^{\odot k} \odot D_2 = D_2 \odot (D_1 D_2)_U^{\odot k} \tag{18}$$

and

$$(D_2 D_1)_U^{\odot k} = D_2 \odot (D_1 D_2)_U^{\odot k-1} \odot D_1 \oplus U_{n_2}. \tag{19}$$

[1] $A_U^{\odot 2} \oplus A = A^{\odot 2} \oplus \underbrace{(A \odot U) \oplus (A \odot U)}_{A} \oplus \underbrace{U^{\odot 2}}_{U} \oplus A = A^{\odot 2} \oplus A \oplus U \oplus A = A \odot (A \oplus U) \oplus (A \oplus U) = (A \oplus U)^{\odot 2} = A_U^{\odot 2}$, *where A is a matrix and U - tropical identiy matrix with proper dimensions*

Proof. The first equation follows from a repeated application of the identity

$$(D_2D_1)_U \odot D_2 = D_2 \odot (D_1D_2)_U. \tag{20}$$

In view of the second equation, we have by (18),

$$\begin{aligned}(D_2D_1)_U^{\odot k} &= (D_2D_1)_U^{\odot k-1} \odot (D_2D_1)_U = (D_2D_1)_U^{\odot k-1} \odot (D_2 \odot D_1 \oplus U_{n_2}) \\ &= ((D_2D_1)_U^{\odot k-1} \odot D_2 \odot D_1) \oplus ((D_2D_1)_U^{\odot k-1} \odot U_{n_2}) \\ &= D_2 \odot (D_1D_2)_U^{\odot k-1} \odot D_1 \oplus (D_2D_1)_U^{\odot k-1}.\end{aligned} \tag{21}$$

Successive employment of this equation provides the identity

$$(D_2D_1)_U^{\odot k} = D_2 \odot \Big[(D_1D_2)_U^{\odot k-1} \oplus \ldots \oplus (D_1D_2)_U \oplus U\Big] \odot D_1 \oplus U_{n_2}. \tag{22}$$

But $(D_1D_2)_U \oplus U = (D_1D_2)_U$ and so we obtain

$$(D_2D_1)_U^{\odot k} = D_2 \odot \Big[(D_1D_2)_U^{\odot k-2} \oplus \ldots \oplus (D_1D_2)_U \oplus U\Big] \odot (D_1D_2)_U \odot D_1 \oplus U_{n_2}. \tag{23}$$

Recurrent invocation yields the desired equality (19). □

Putting both propositions together yields the following result.

Corollary 3.
For any integer $k \geqslant 1$, the distance matrix has the form

$$D^{\odot 2n_1} = \begin{pmatrix} (D_1D_2)_U^{\odot n_1} & (D_1D_2)_U^{\odot n_1-1} \odot D_1 \\ D_2 \odot (D_1D_2)_U^{\odot n_1-1} & D_2 \odot (D_1D_2)_U^{\odot n_1-1} \odot D_1 \oplus U_{n_2} \end{pmatrix}. \tag{24}$$

Note that the four blocks of the distance matrix $D^{\odot 2n_1}$ depend on the $n_1 \times n_1$ matrix

$$C = (D_1D_2)_U^{\odot n_1-1}. \tag{25}$$

This observation leads to a straightforward APSP algorithm for calculating the distance matrix (Procedure 1 APSP-Bigraph). For this, the matrix C will be precalculated and then the four blocks will be computed as in the corollary.

4. Algorithm

The implementation of the Procedure 1 APSP-Bigraph could be improved by calculating matrix powers by repeated squaring, referred further on as APSP-Bigraph(RS). This algorithm enhances the ordinary APSP procedure based on the distance matrix product (DistanceMatrixProduct(RS)) in two ways. First, its time complexity is $O(\ell(n_1)\log_2 n_1)$ compared to the runtime $O(\ell(n)\log_2 n)$ of the basic method. Here $O(\ell(n))$ denotes the complexity for the multiplication of $n \times n$ matrices. If ordinary matrix multiplication is used, i.e., $\ell(n) = n^3$, then in the worst case, i.e. with $n_1 = n_2$, the matrix exponentiation by APSP-Bigraph(RS) is eight times faster, than that of the DistanceMatrixProduct(RS). Since, if $n_1 = n_2$, then $n_1 = n/2$, and matrix multiplication takes $n_1^3 = (\frac{n}{2})^3 = \frac{1}{8}n^3$ steps. Second, the dimension of the exponentiated matrix is reduced from $(n_1 + n_2)^2$ to n_1^2. Thus, in the worst case, the respective space required by the algorithm APSP-Bigraph(RS) is four times smaller than that of the DistanceMatrixProduct(RS). Note that the advantage of APSP-Bigraph(RS) becomes more significant for graphs with large ratio of n/n_1 such as client-server computer networks.

Procedure 1 APSP-Bigraph

Input: cost matrix D given by the blocks D_1 and D_2
Output: distance matrix $D^{\odot 2n_1}$ specified by the blocks $L_1^{(2n_1)}$, $L_2^{(2n_1)}$, $D_1^{(2n_1)}$, and $D_2^{(2n_1)}$

$C = U$
$C_1 = D_1 \odot D_2 \oplus U$

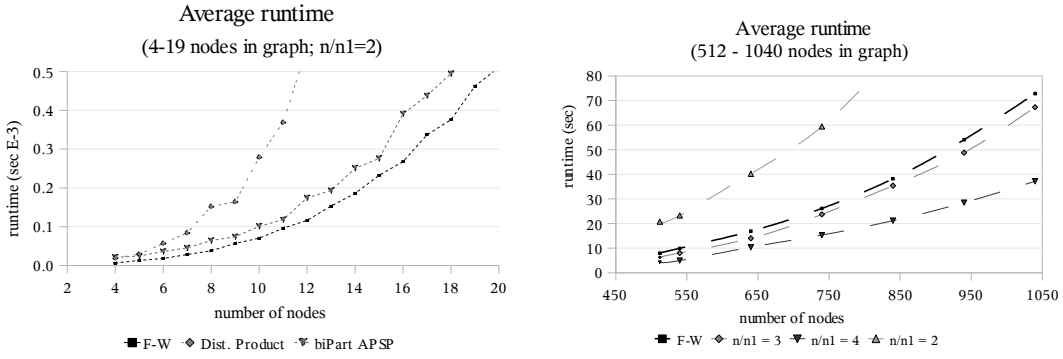
/* *calculate* $(D_1 D_2)_U^{\odot n_1 - 1}$ */

for $k = 1$ to $n_1 - 1$ **do**
  $C = C \odot C_1$
end for

/* *compute the blocks of the distance matrix* */

$L_1^{(2n_1)} = C \odot C_1$
$L_2^{(2n_1)} = D_2 \odot C \odot D_1 \oplus U$
$D_1^{(2n_1)} = C \odot D_1$
$D_2^{(2n_1)} = D_2 \odot C$

return $L_1^{(2n_1)}$, $L_2^{(2n_1)}$, $D_1^{(2n_1)}$, $D_2^{(2n_1)}$



(a) Small complete graphs with $n/n_1 = 2$.

(b) Complete graphs with $n/n_1 = 2, 3, 4$.

Figure 2. Runtime comparison. (a) Runtime comparison between three APSP methods for small complete bipartite graphs. The upper curve provides the runtime of DistanceMatrixProduct(RS), the middle one – of APSP-Bigraph(RS), and the lower one – of Floyd-Warshall. Time of APSP-Bigraph(RS) is measured for the worst case graphs ($n/n_1 = 2$). (b) Runtime for bipartite graphs with 512 to 1040 nodes and with different n/n_1 ratio. Dashed curves show runtime of APSP-Bigraph(RS) for graphs with n/n_1 ratio of 2, 3, and 4 from top to bottom, respectively. Solid line represents the runtime of Floyd-Warshall, which is independent from the value of n/n_1.

Runtime comparison

The runtime of the proposed algorithm APSP-Bigraph(RS) was compaired with that of two other matrix-based APSP methods: Floyd-Warshall and DistanceMatrixProduct(RS) (Table 1 and Figure 2). The performance was measured on a Pentium DoubleCore processor with 1.6 GHz. The results are based on 50 randomly generated bipartite graphs of each graph order n.
Table 1 illustrates the runtime of three methods for complete and sparse(in brackets) bipartite graphs: DistanceMatrixProduct(RS), APSP-Bigraph(RS), and Floyd-Warshall. APSP-Bigraph(RS) is significantly faster than DistanceMatrixProduct(RS), but in the worst case ($n_1 = n/2$) it is outperformed by Floyd-Warshall. On the other hand, Figure 2(b) exhibits that APSP-Bigraph(RS) outperforms Floyd-Warshall by the values of the ratio $n/n1$ higher than or equal to three.

5. Discussion

The class of bipartite graphs is used in a wide range of practical applications such as transportation systems, robotics, and networking. Modelling of such systems often requires a division of their entities into two mutually unconnected

Table 1. Runtime of three APSP algorithms.

Method	*Complexity*	*Graph order (n)*						
		2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
		average runtime(sec)						
		$\times 10^{-4}$	$\times 10^{-3}$	$\times 10^{-3}$	$\times 10^{-1}$			
Floyd-Warshall	n^3	3.3 (1.1)	2.11 (1.13)	16.4 (9.5)	1.21 (0.95)	1.02 (0.81)	7.95 (6.65)	60.98 (50.7)
DistanceMatrixProduct(RS)	$n^3 \log_2 n$	20 (8.1)	16.4 (10.8)	161.1 (119)	16.0 (13.9)	15.79 (13.7)	153.05 (145.3)	1605.43 (1360)
APSP-Bigraph(RS) ($n/n_1 = 2$)	${n_1}^3 \log_2 n_1$	5.3 (2.3)	3.4 (2.00)	30.1 (18.5)	2.67 (2.11)	2.55 (2.01)	21.88 (19.7)	216.95 (160.5)
		speedup(times)						
APSP-Bigraph(RS) vs. DistanceMatrixProduct(RS)		3.8	4.8	5.4	6	6.2	7	7.4
APSP-Bigraph(RS) vs. Floyd-Warshall		0.6	0.6	0.5	0.5	0.4	0.36	0.28

(RS) - repeated squaring matrix multiplication; $n_1 = |V_1|$

classes. Examples are popular models such as Petri nets, neural networks, electrical circuits, workflows, and trees.

The proposed algorithm comprises the general case of directed weighted bipartite graphs, unlike the other existing methods for this class of graphs, such as from Chen [6] and Chin-Wen and Chang [11], which handle more specific situations like restrictions of edges costs or topology. In this way, the area of applications of the presented method is significantly larger.

The proposed method is also of practical importance, since the calculations are based on a matrix with reduced dimensionality. The theoretical complexity remains the same as of the distance product APSP with repeated matrix squaring: $O(|V|^3 \log_2 |V|)$. Further acceleration of matrix multiplication seems unrealistic. The respective methods, such as ones of Strassen [16] or Coppersmith and Winograd [17], are inapplicable for obtaining the distance product, since they imply an operation inverse to addition. Another approach, proposed by Fredman [4], would be expensive to implement for larger matrices [10].

The size of the exponentiated matrix amounts to one-fourth of the size of the cost matrix, whereas some of the alternative methods for bipartite graphs (e.g., Chin-Wen and Chang [11]) achieve their acceleration on the cost of additional space proportional to the size of the cost matrix.

The reduced dimensionality of the cost matrix is highly welcome in parallel computations on special hardware such as graphic processor units. In such systems, the APSP calculations are limited to graphs with several thousands of vertices due to the memory capacity [18, 19]. For such applications, the smaller matrix allows to extend this bound and to compute the APSP for graphs, which are out of range for the other known methods on such hardware.

For large-scale applications based on bipartite graphs, such as the one cited at the beginning of this section, the achieved space and time reduction makes the presented method highly useful as a subroutine. Another advantage is its straightforward implementation as documented by the pseudocode.

6. Conclusion

The presented APSP method is applicable to directed and undirected real-weighted bipartite graphs without negative cycles. In the worst case, it reduces the size of matrices used in the computation of the distance matrix to one-fourth of the size of the original cost matrix. Moreover, the runtime in the worst case becomes at least seven times faster for large graphs, when compared with the distance matrix product APSP method. Furthermore, a higher gain can be achieved when the ratio n/n_1 becomes large. So, for the n/n_1 ratio higher than or equal to three, the presented method outperforms the Floyd-Warshall algorithm (Fig. 4). The proposed method also offers a significant advantage for large-scale applications. Indeed, the presented reduction of space complexity is highly useful in parallel computations on special hardware such as graphic processor units [18, 19].

References

[1] R. W. Floyd, Algorithm 97: Shortest path, Commun. ACM. 5(6), 345, 1962
[2] S. Warshall, A theorem on boolean matrices, J. ACM, 9(1), 11, 1962
[3] D. B. Johnson, Efficient algorithms for shortest paths in sparse networks, J. ACM, 24(1), 1, 1977
[4] M. L. Fredman, New bounds on the complexity of the shortest path problem, SIAM J. Comput. 5, 83, 1976
[5] W. Dobosiewicz, A more efficient algorithm for the min-plus multiplication, Int. J. Comput. Math. 32(1), 49, 1990
[6] L. Chen, Solving the shortest-paths problem on bipartite permutation graphs efficiently, Inform. Process. Lett. 55(5), 259, 1995
[7] T. Takaoka, Sub-cubic Cost Algorithms for the All Pairs Shortest Path Problem, Algorithmica 20, 309, 1995
[8] D. Dor, S. Halperin, U. Zwick, All-pairs almost shortest paths, SIAM J. Comput. 29, 1740, 2000
[9] F. F. Dragan, Estimating all pairs shortest paths in restricted graph families: a unified approach, J. Algorithm. 57(1), 1, 2005
[10] U. Zwick, A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths, Algorithmica 46(2), 181, 2006
[11] H. Chin–Wen, J. M. Chang, Solving the all-pairs-shortest-length problem on chordal bipartite graphs, Inform. Process. Lett. 69(2), 87, 1999
[12] A. Shimbel, Structure in communication nets. Proceedings of the Symposium on Information Networks, Vol. 4 (New York, 1954, Polytechnic Press of the Polytechnic Institute of Brooklyn, NY, 1955) 199
[13] M. Leyzorek, R. S. Gray, A. A. Johnson, W. C. Ladew, S. R. Meaker Jr, R. M. Petry, R. N. Seitz, Investigation of Model Techniques — First Annual Report — A Study of Model Techniques for Communication Systems, (Case Institute of Technology, Cleveland, Ohio, 1957)
[14] I. Simon, In: M. P. Chytill (Ed.), Recognizable sets with multiplicities in the tropical semiring, Mathematical Foundations of Computer Science 1988, Proc. MFCS'88, Lecture Notes in Computer Science Vol. 324 (Springer, Berlin, 1988) 107–120
[15] D. Speyer, B. Sturmfels, Tropical Mathematics, arXiv/math/0408099v1
[16] V. Strassen, Gaussian elimination is not optimal, Numerische Mathematik, 13(4), 354, 1969
[17] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, J. Symb. Comput. 9(3), 251, 1990
[18] P. Micikevicius, General parallel computation on commodity graphics hardware: Case study with the all-pairs shortest paths problem, PDPTA, Citeseer, 4, 1359, 2004
[19] P. Harish, P. Narayanan, Accelerating large graph algorithms on the GPU using CUDA, High Perform. Comput.–HiPC 2007, 197, 2007