Increasing Parallelism in Forward-backward Distributed Algorithm for Finding Strongly Connected Components of Directed Graphs

Dominik Ryżko

Warsaw University of Technology, Warsaw, Poland

https://doi.org/10.26636/jtit.2024.3.1693

Abstract — The paper proposes a modification of the existing distributed forward-backward algorithm for finding strongly connected components in directed graphs. The modification aims at improving the parallelism of the algorithm by increasing the branching factor while dividing the workload. Instead of randomly picking the pivot vertex, a heuristic technique is used which allows more sub-tasks to be generated, on average, for the subsequent step of the algorithm. The work describes suitable algorithm modifications and presents empirical results, proving suitability of the approach in question.

Keywords — distributed computation, graph algorithms, strongly connected components

1. Introduction

Directed graphs are an elegant formalism suitable for representing several abstract notions and natural phenomena. For real life problems, such graphs contain a lot of detailed information, so their size becomes extremely large. Therefore, there is a need to decompose such structures into meaningful sub-graphs which can be interpreted and processed independently.

One of the most important potential decompositions is the detection of strongly connected components (SCCs), i.e. maximal subsets of vertices in which a directed path exists from each vertex to any other vertex.

SCC detection has several practical applications, including in telecommunications [1], [2], radiation transport solvers [3], distributed formal verification [4], distributed reasoning [5], [6] analysis of citation graphs [7], [8] page rank calculations [9], and social networks [10]. The number, size and ordering of SCCs may provide valuable information about the data represented by the graph. Consequently, one may gain a deeper insight into the specific phenomenon modeled by the data.

Various SCC detection algorithms have been proposed in literature so far. Along with the growing availability of parallel computation architectures, scientific interest has shifted recently towards those algorithms that are able to effectively utilizing such capabilities. In order to take advantage of the efficiency of distributed processors, the algorithms have to split the workload into several sub-tasks which can be computed separately. The degree of such decomposition can be measured by the branching factor of a tree of tasks generated at each step of a recursive algorithm.

In the case of several parallel algorithms used for finding SCCs, selection of a pivot, i.e. vertex from which the decomposition begins, is an important step impacting the algorithm branching process. Typically, a random vertex is selected, with such an approach usually rendering satisfactory results.

In this work, we introduce a heuristic pivot selection method improving parallelism by increasing the average branching factor of the algorithms.

The remainder of the paper is structured as follows. Section 2 describes state of the art in the field of SCC detection, with the emphasis placed on parallel algorithms and pivot selection strategies. In Section 3, basic concepts used in the article are defined. A description of heuristics and algorithm modifications is presented in Section 4. Section 5 presents empirical results from experiments conducted with the use of the algorithms in question. Finally, conclusions and future work are presented in Section 6.

2. Related Work

To date, several algorithms for finding SCCs have been proposed. Classical approaches based on depth-first search [11], [12] have been proven to work in linear time with respect to the number of edges. This group of algorithms is difficult to parallelize and, therefore, is not useful in multi-core architectures. However, some limited results have been achieved in this area [13]. Also, several distributed algorithms for speeding up the process have been proposed and successfully refined [14]–[18]. The most important of these algorithms will be described in the subsequent sections.

With the proliferation of GPUs observed in recent years, multiple algorithms rely on their computational power to improve efficiency and inherent parallelism [19], [20].

2.1. Forward-backward Algorithm

One of the most popular approaches has the form of the forward-backward (FW-BW) algorithm proposed originally by Fleischer *et al.* [21] and further refined by other researchers [3], [22]. It is based on the computation of forward and backward closures of a selected node, referred to as a pivot. One can prove that the intersection of the closures is a single SCC and that the algorithm may be recursively called for three disjointed sets, i.e. vertices in the forward closure, but not in the backward closure, vertices in the backward closure but not in the forward closure, and the remaining vertices which did not belong to any of the closures.

Pivot selection is an important step in the above algorithm, since it exerts a direct impact on how many sub-problems can be generated in the subsequent step. To date, in most of the approaches random pivot detection is used. Several other methods have been proposed as well.

Barnat and Moravec [23] introduced an approach to prevent some trivial strongly connected components from being selected as pivots. In this method, the pivot is chosen out of a set of candidates computed with the use of the back-level edge concept. A back-level edge in a cycle is an edge that leads from a vertex with some distance from the initial vertex to a vertex with an equal or smaller distance from the initial vertex. The solution is to use destination vertices of the back-level edges as candidate pivots.

Another approach to pivot selection can be found in [18], where a multipivot algorithm is proposed. The idea is to sample s vertices and to start the reachability procedure simultaneously for each of them. Once this is done, the following sets are computed:

$A =$ vertices reached from vertex set $\{1, 2, ..., s-1\}$,

$B =$ vertices reached from vertex s,

$C =$ vertices that reach and are reached from s.

Finally, we can recursively compute, in parallel, the strongly connected components of:
$V - (A \cup B)$, $A - B$, $B - (A \cup C)$, and $(A \cap B) - C$.

This paper proposes yet another heuristic pivot selection technique. The advantage of the proposed approach is that it involves practically no computational or memory overhead.

Alternative improvements to the FW-BW approach have been proposed using the idea that a spanning tree can be employed instead of the breadth first search (BFS) tree. A promising version of such an approach has been shown by Ji *et al.* [24], with a prototype called iSpan, consisting of a parallel, relaxed synchronization design of spanning trees for detecting large and small SCCs, combined with fast trims for small SCCs.

The trimming phase is an important improvement to the FW-BW algorithm and is a subject of a separate field of research seeking further improvements (e.g. [25]), where FB-AI-Trim, a parallel SCC algorithm with selective trimming is proposed.

2.2. OBF Algorithm

OBF [23] is another parallel SCC detection algorithm which identifies a number of independent sub-graphs (called OBF slices) in $O(n + m)$ time. The slices are then decomposed using the FW-BW algorithm. The algorithm starts with a complete set of vertices and selection of the initial vertex.

One OBF slice is computed is one step. Then, the identified slice is decomposed and the remaining part of the graph is analyzed, in parallel, for extraction of the next slice. The OBF algorithm has been improved to create the recursive OBF approach, in which OBF is applied recursively to the extracted slices [26].

2.3. Coloring Algorithm

Yet another approach to SCC detection is based on the concept of coloring [27]. Such a technique starts with a set of ordered colors. Initially, all vertices are assigned different colors. Then, they are propagated along the edges, so that a higher color overwrites a smaller one. As the result, all vertices in a single SCC have the same color. Therefore, the edges between different colors can be removed and the resulting sub-graphs can be processed independently.

In contrast to the FW-BW algorithm, the coloring algorithm allows to split the graph into more parts. However, this implies a higher overhead.

2.4. Graph Generation

This work shows that the process of generating a graph to test the performance of an algorithm is of crucial significance. Ideally, an approach would be preferred which outputs uniformly random graphs while controlling the key parameters of the graph, i.e. its size, density, number of SCCs etc.

$G(n, p)$ and $G(n, M)$, as proposed by Erdos and Renyi [28], are some of the most widely used graph generation methods which can be harnessed for non-oriented graphs, as well as DAGs. Other methods include: layer-by-layer [29], fan-in/fan-out [30], and random orders [31].

While these algorithms provide some means for predefining general graph properties, such as the number of vertices and edges, for the case described in this work we also wanted to control other parameters, with a particular emphasis placed on the number of SCCs. To this end, a custom graph generation method has been employed.

3. Basic Concepts

We start with the definition of a directed graph:

Definition 1. A *directed graph* G is a pair (V, E), where V is a set of vertices, and $E \subseteq V \times V$ is a set of directed edges. If $(u, v) \in E$, then v is called immediate successor of u, and u is called immediate predecessor of v.

Definition 2. A *directed path* in a digraph is a sequence of vertices in which there is a (directed) edge pointing from each vertex in the sequence towards its successor in the sequence. A simple path is the one with no repeated vertices.

A *strongly connected component* (SCC) of a directed graph G is defined in the following way:

Algorithm 1 DetectSCCs

```
1:  if $G$ has no more then one vertex then return
2:      TRIM $G$ in forward direction
3:      if $G$ is not empty then
4:          TRIM $G$ in backward direction
5:          select vertex $v$ from $G$
6:          MARK $Pred\,(G,v)$ and $Desc\,(G,v)$ in $G$
7:          $SCC(G,v) = Pred\,(G,v) \cap Desc\,(G,v)$
8:          Do in parallel
9:          $detectSCCs(Pred\,(G,v) - SCC(G,v))$
10:         $detectSCCs(Desc\,(G,v) - SCC(G,v))$
11:         $detectSCCs(Rem\,(G,v))$
12:     end if
13: end if
```

Definition 3. A *strongly connected component* of a directed graph G is a maximal set of vertices $C \subseteq V$ such that for every pair of vertices u and v, there is a directed path from u to v and a directed path from v to u.

A back-level edge is a concept used by one of the pivot selection algorithms presented in Section 2.

Definition 4. Let $G = (V, E)$ be a graph with the source vertex s. An edge $(u, v) \in E$ is called the *back-level edge* if and only if $d(u) >= d(v)$, where $d(u)$ and $d(v)$ are the distances from s to u and s to v, respectively. Vertex u is called the start vertex of the back-level edge, while v is called destination vertex.

4. Increasing Parallelism in FW-BW SCC Detection Algorithm

This section presents improvements to the SCC detection algorithms. We start with a description of the standard FW-BW algorithm and then discuss the pivot selection problem.

4.1. FW-BW Algorithm

Let us assume that we intend to discover all strongly connected components in a graph by means of computations performed asynchronously by a set of nodes. Following the work performed by McLendon [3], we can apply, to this end, a distributed algorithm performed at each node participating in the process (Algorithm 1).

The trim operations are performed in order to remove those vertices which do not belong to the SCC. The forward trim begins with all vertices that have no ancestors and removes them together with all their edges. After the removal, some other vertices may have no ancestors as well, and they are removed too. The process continues until no more vertices can be removed. More formally, the trim iteratively removes vertices whose in-degree is non-zero and whose out-degree is zero.

By analogy, the reverse trim performs the same operation in the other direction.

4.2. Pivot Selection

The problem of pivot selection is important for the described algorithm. Ideally, such a pivot should be picked each time that produces four sub-graphs at the end of the algorithm step, allowing for the highest possible branching of the recursion. However, if no ancestors or predecessors exist (or there are a few of them and all belong to the intersection of closures) for the particular SCC detected at the current step, then the branching factor will be smaller and fewer sub-processes will be started.

In most of the algorithms proposed so far, the pivot was selected randomly. The argument was that the optimal pivot selection process requires depth-first search post-ordering, which is rather difficult to perform in parallel. In this paper, we bring forward the observation that a useful heuristic may be applied by using the information already computed as one of the algorithm steps.

First, the reachability procedure is performed for two out of three sets generated by the FW-BW algorithm. This returns us the breadth first partial ordering over the vertices in these sub-graphs. This information can be used to avoid selecting, as pivots, those vertices which are likely to belong to an SCC with no descendants or predecessors. These will most likely be the vertices reached at the beginning or at the end of the process.

Let us analyze the example presented in Fig. 1. The sub-graph represents all vertices reachable from v_2. The possible order of visiting the vertices by the breadth-first search algorithm is shown in Fig. 2.

Since some of these vertices will belong to the SCC (darker color), they will be excluded from the set passed on to the next algorithm step, meaning they should not be considered for the pivot selection process.

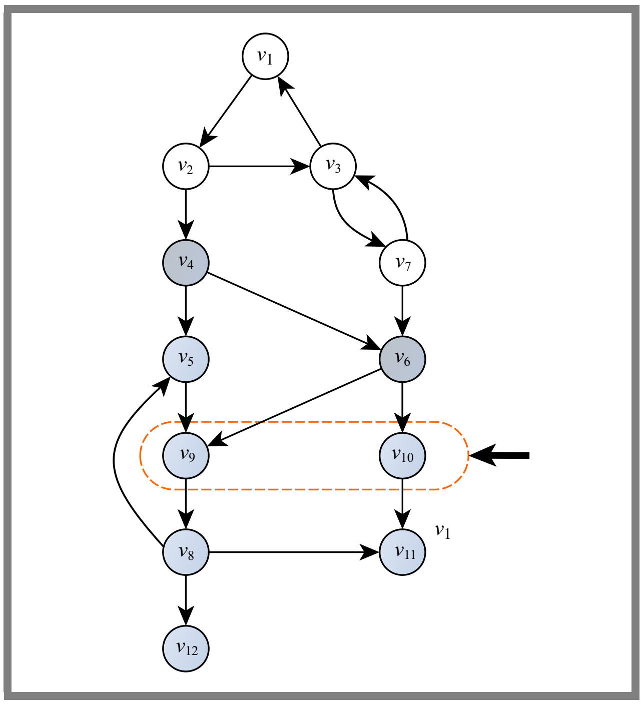

Fig. 1. Example graph.

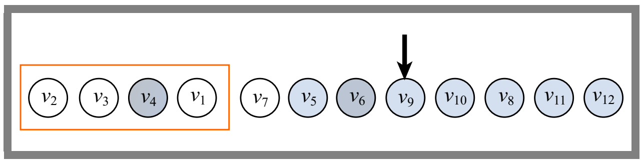

Fig. 2. Order of vertices visited by the breadth-first search algorithm.

Vertices located in the middle of the graph (after exclusion of SCC vertices), after breadth-first search layering, are most likely to have a sizable set of predecessors and ancestors. This layer is marked in Fig. 1. On the other hand, picking a vertex from the top or bottom layers (e.g. v_4 or v_{12}) leaves us with one sub-task empty.

Following the observation described above, we should ideally compute the middle layer of the relevant sub-graph and pick a vertex out of it. However, since a precise identification of these vertices can introduce some computational overhead, we have tested a simplified solution.

We assign sequential numbers to the vertices during the reachability procedure and then the index of the pivot is calculated as:

$$idx = \big(size(ForwardClosure) - size(SCC)\big) \text{ div } 2$$

or

$$idx = \big(size(BackwardClosure) - size(SCC)\big) \text{ div } 2,$$

respectively.

If the index points to a vertex which belongs to the SCC, which is possible with such a simplification, a random vertex is chosen.

Algorithm 2 presents the final version of the SCC detection including the improvements described above.

5. Experimental Results

To test the usability of the heuristic technique described in the Section 4, a specific test environment has been created. The standard FW-BW algorithm has been implemented using the Scala Akka library for parallel computations [32]. Then, the modification of pivot selection has been added. The test environment has been run on a multi-core workstation. It is important to note that the branching factor can be measured independently of the physical computational environment. It impacts the execution times only.

5.1. Graph Generation

To analyze the performance of the algorithm on graphs with different numbers and sizes of the SCCs, a dedicated method for test graph generation has been adopted. First, a spanning tree is generated over the initial set of vertices. Then, the vertices are divided equally (if possible) into a predefined number of sets and a predefined number of vertices is added within each set only. Finally, some extra edges are added between the sets.

Such an approach allows us to control the graph size, its density, as well as the number and size of SCCs. Due to the random character of some of the generation algorithm steps, some of the graph parameters, e.g. number of SCCs, may vary when given the same algorithm parameters. However, these variations are very small and do not affect the results of the experiments.

5.2. Experiments

In order to test the modified algorithm, it has been run for graphs of various sizes and SCC counts. It has been benchmarked against the standard algorithm. Figure 3 shows a comparison of the average branching factor for a graph containing 100k vertices, 400k edges and various SCC counts.

The scalability of the new algorithm that may be adapted to the growing graph size has been tested and is shown in Fig. 4.

Figure 5 shows the run time (in log scale) as a function of the graph size (vertices/SCCs), when run on a 4-core computer system. While the branching factor is independent of the environment, the run time still shows some improvement potential when a system with more cores is used. But the improvement is visible even on a 4-core system.

5.3. Experiment Outcomes

Next, the algorithm has been run for various graph sizes and various number of SCCs within the graph. The results prove the initial assumption that it is capable of increasing the branching factor. It scales according to graph size and the number of SCCs for a fixed number of vertices. As practically no additional computational overhead is present, the execution times are also consistently better.

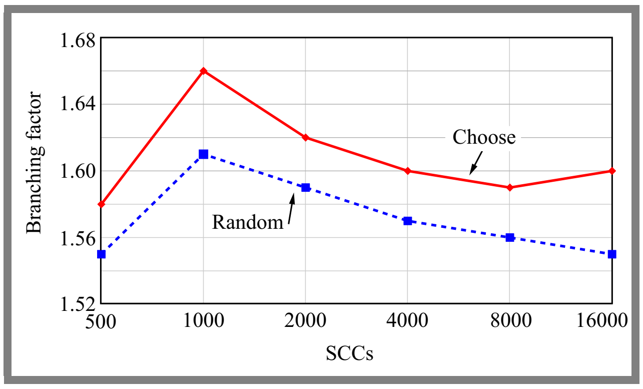

Fig. 3. Branching factor as a function of the number of SCCs.

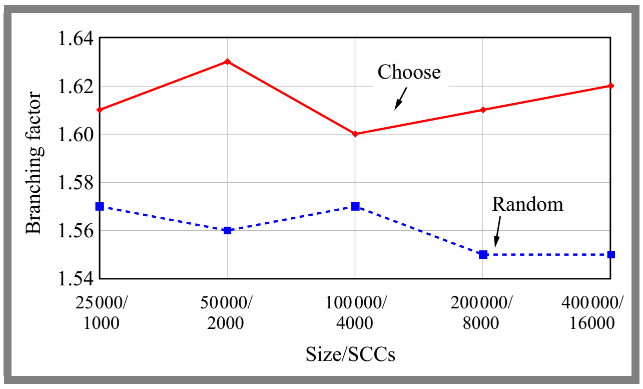

Fig. 4. Branching factor as a function of the size of the graph.

Algorithm 2 ImprovedDetectSCCs

```
Input: $pivot$
if $G$ has no more then one vertex then return
    TRIM $G$ in forward direction
    if $G$ is not empty then
        TRIM $G$ in backward direction
        if $pivot = \emptyset$ then select random vertex $v$ from $G$
            MARK $Pred\,(G, v)$ and $Desc\,(G, v)$ in $G$
            $SCC(G, v) = Pred\,(G, v) \cap Desc\,(G, v)$
            $FwdPivot = Desc\left[\left(size(Desc) - size(SCC)\right) \text{ div } 2\right]$
            if $FwdPivot \in SCC$ then $FwdPivot = \emptyset$
                $BckPivot = Pred\left[\left(size(Pred) - size(SCC)\right) \text{ div } 2\right]$
                if $BckPivot \in SCC$ then $BckPivot = \emptyset$
                    Do in parallel
                    $detectSCCs\,(Pred(G, v) - SCC(G, v), BckPivot)$
                    $detectSCCs\,(Desc(G, v) - SCC(G, v), FwdPivot)$
                    $detectSCCs\,(Rem(G, v))$
                end if
            end if
        end if
    end if
end if
```

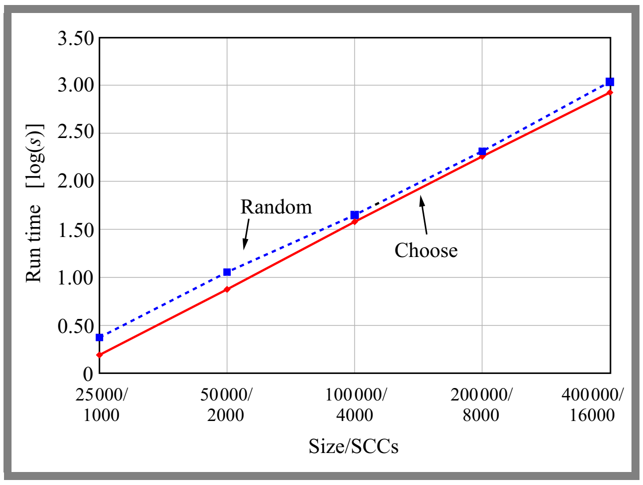

Fig. 5. Run time as a function of the size of the graph.

6. Conclusions

This paper presents a potential improvement to the pivot selection process being part of the FW-BW algorithm used for SCC decomposition. It has been shown that the branching factor of the algorithm may be increased based on the information already computed in the previous algorithm step, hence without introducing significant overhead.

Future work should involve blending this approach with other pivot selection techniques, such as the multipivot or degree-based methods which are also mentioned herein. It could be also interesting to find out if the heuristic introduced here may be relied upon to solve other problems involving pivot selection e.g. clique enumeration etc.

References

[1] B. Sandeep and A. Ferreira, "Complexity of Connected Components in Evolving Graphs and the Computation of Multicast Trees in Dynamic Networks", *Second International Conference, ADHOC-NOW2003*, Montreal, Canada, 2003 (https://doi.org/10.1007/978-3-540-39611-6_23).

[2] S. Raghavan, "Twinless Strongly Connected Components", in: *Perspectives in Operations Research*, pp. 285–304, 2006 (https://doi.org/10.1007/978-0-387-39934-8_17).

[3] W. McLendon III, B. Hendrickson, S.J. Plimpton, and L. Rauchwerger, "Finding Strongly Connected Components in Distributed Graphs", *Journal of Parallel and Distributed Computing*, vol. 65, no. 8, pp. 901–910, 2005 (https://doi.org/10.1016/j.jpdc.2005.03.007).

[4] H. Garavel, R. Mateescu, and I.M. Smarandache, "Parallel State Space Construction for Model-checking", *Proc. of the 8th International SPIN Workshop on Model Checking of Software (SPIN'01)*, vol. 2057, pp. 200–216, 2001 (https://doi.org/10.1007/3-540-45139-0_14).

[5] D. Ryzko, "Reasoning in Multi-agent Systems with Random Knowledge Distribution", in: *Advances in Databases and Information Systems*, pp. 310–317, 2012 (https://doi.org/10.1007/978-3-642-33074-2_23).

[6] P. Cholewinski, "Reasoning with Stratified Default Theories", *Proc. of Third International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR'95)*, pp. 273–286, 1995 (https://dl.acm.org/doi/10.5555/646397.691152).

[7] V. Batagelj, "Efficient Algorithms for Citation Network Analysis", *ArXiv*, 2003 (https://doi.org/10.48550/arXiv.cs/0309023).

[8] Y. Kajikawa *et al.*, "Creating an Academic Landscape of Sustainability Science: An Analysis of the Citation Network", *Sustainability Science*, vol. 2, pp. 221–231, 2007 (https://doi.org/0.1007/s11625-007-0027-8).

[9] H. Yang *et al.*, "Strongly Connected Components Based Efficient PPR Algorithms", *Jisuanji Xuebao/Chinese Journal of Computers*, vol. 40, pp. 584–600, 2017.

[10] N.P. Hummon and P. Doreian, "Computational Methods for Social Network Analysis", *Social Networks*, vol. 12, no. 4, pp. 273–288, 1990 (https://doi.org/10.1016/0378-8733(90)90011-W).

[11] R. Tarjan, "Depth First Search and Linear Graph Algorithms", *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146–160, 1972 (https://doi.org/10.1137/0201010).

[12] H.N. Gabow, "Path-based Depth First Search Strong and Biconnected Components", *Information Processing Letters*, vol. 74, no. 3–4, pp. 107–114, 2000 (https://doi.org/10.1016/S0020-0190(00)00051-X).

[13] E. Renault, A. Duret-Lutz, F. Kordon, and D. Poitrenaud, "Parallel Explicit Model Checking for Generalized Buchi Automata", in: *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 613–627, 2015 (https://doi.org/10.1007/978-3-662-46681-0_56).

[14] H. Gazit and G.L. Miller, "An Improved Parallel Algorithm that Computes the BFS Numbering of a Directed Graph", *Information Processing Letters*, vol. 28, no. 2, pp. 61–65, 1988 (https://doi.org/10.1016/0020-0190(88)90164-0).

[15] R. Cole and U. Vishkin, "Faster Optimal Parallel Prefix Sums and List Ranking", *Information and Computation*, vol. 81, no. 3, pp. 334–352, 1989 (https://doi.org/10.1016/0890-5401(89)90036-9).

[16] J. Barnat, J. Chaloupka, and J. van de Pol, "Improved Distributed Algorithms for SCC Decomposition", *Electronic Notes in Theoretical Computer Science*, vol. 198, no. 1, pp. 63–77, 2008 (https://doi.org/10.1016/j.entcs.2008.02.001).

[17] W. Schudy, "Randomized Algorithms for Graph Problems", 2007.

[18] W. Schudy, "Finding Strongly Connected Components in Parallel Using $O(\log^2 n)$ Reachability Queries", *Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures*, pp. 146–151, 2008 (https://doi.org/10.1145/1378533.1378560).

[19] J. Jaiganesh and M. Burtscher, "A High-performance Connected Components Implementation for GPUs", *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 92–104, 2018 (https://doi.org/10.1145/3208040.3208041).

[20] T. Ben-Nun, M. Sutton, S. Pai, and K. Pingali, "Groute: An Asynchronous Multi-GPU Programming Model for Irregular Computations", *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 235–248, 2017 (https://doi.org/10.1145/3018743.3018756).

[21] L. Fleischer, B. Hendrickson, and A. Pinar, "On Identifying Strongly Connected Components in Parallel", *Proceedings of Parallel and Distributed Processing Symposium*, pp. 505–511, 2000 (https://doi.org/10.1007/3-540-45591-4_68).

[22] S. Hong, N.C. Rodia, and K. Olukotun, "On Fast Parallel Detection of Strongly Connected Components (SCC) in Small-world Graphs", *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013 (https://doi.org/10.1145/2503210.2503246).

[23] J. Barnat and P. Moravec, "Parallel Algorithms for Finding SCCs in Implicitly Given Graphs", in: *Formal Methods: Applications and Technology*, pp. 316–330, 2007 (https://doi.org/10.1007/978-3-540-70952-7_22).

[24] Y. Ji, H. Liu, and H.H. Huang, "iSpan: Parallel Identification of Strongly Connected Components with Spanning Trees", *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, Dallas, USA, 2022 (https://doi.org/10.1109/SC.2018.00061).

[25] D. Niewenhuis and A.L. Varbanescu, "Efficient Trimming for Strongly Connected Components Calculation", *Proceedings of the 19th ACM International Conference on Computing Frontiers*, pp. 131–140, 2022 (https://doi.org/10.1145/3528416.3530247).

[26] J. Barnat, J. Chaloupka, and J. van de Pol, "Distributed Algorithms for SCC Decomposition", *Journal of Logic and Computation*, vol. 21, no. 1, pp. 23–44, 2011 (https://doi.org/10.1093/logcom/exp003).

[27] S. Orzan, "On Distributed Verification and Verified Distribution", Ph.D. thesis, Free University of Amsterdam, 2004 (https://research.vu.nl/en/publications/on-distributed-verification-and-verified-distribution).

[28] P. Erdos and A. Renyi, "On Random Graphs I", *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, 1959

[29] T. Tobita and H. Kasahara, "A Standard Task Graph Set for Fair Evaluation of Multiprocessor Scheduling Algorithms", *Journal of Scheduling*, vol. 5, no. 5, pp. 379–394, 2002 (https://doi.org/10.1002/jos.116).

[30] R.P. Dick, D.L. Rhodes, and W. Wolf, "TGFF: Task Graphs for Free", *Proceedings of the 6th International Workshop on Hardware/Software Codesign*, pp. 97–101, 1998 (https://doi.org/10.1109/HSC.1998.666245).

[31] P. Winkler, "Random Orders", *Order*, vol. 1, pp. 317–331, 1985 (https://doi.org/10.1007/BF00582738).

[32] Akka, (http://akka.io).

Dominik Ryżko, Ph.D.
Institute of Computer Science
https://orcid.org/0000-0001-5632-0861
E-mail: dominik.ryzko@pw.edu.pl
Warsaw University of Technology, Warsaw, Poland
https://eng.pw.edu.pl/