Multi-agent Information System Design and Implementation: Empirical Analysis of IS Subsystems Execution and Development Order Algorithm

Robert Kudelić
University of Zagreb/Faculty of Organization and Informatics,
Pavlinska 2, 42000 Varaždin, Croatia
Email: robert.kudelic@foi.hr

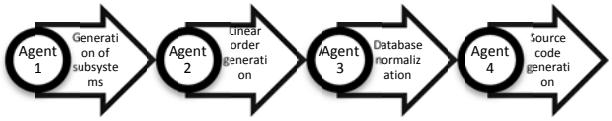
Mladen Konecki and Alen Lovrenčić
University of Zagreb/Faculty of Organization and Informatics, Pavlinska 2, 42000 Varaždin, Croatia
Email: {mladen.konecki, alen.lovrencic}@foi.hr

***Abstract*— Designing an information system demands intense efforts, although a lot of operations that are involved have a potential to be automated or semiautomated. In our previous work we developed advanced agents for automatic determination of IS subsystems through k-way cuts and automatic determination of IS subsystems execution order through an evolutionary approach. Another development was also achieved in automatic database normalization with a view to facilitate and speed up IS design. In this paper we conduct empirical analysis of the algorithm for automatic determination of IS subsystems execution order to establish what its limitations are, whether it behaves satisfactorily and what further improvements are possible to ensure its wider application.**

***Index Terms*—information system, agent, algorithm, linear order, automatic, empirical analysis**

I. Introduction

Information system (IS) design is performed by means of case tools that have seen major advancements although room for their improvement remains. A considerable number of procedures depend on designers' manual effort, while the tool is only used for monitoring results. We therefore decided to automate the entire procedure of IS design that would require minimal corrections on the designer part or at least reduce the amount of work to be executed wherever possible. Our intention was to develop an agent or a series of agents that would intelligently search the solution space. In other words, considering that the entire solution space consists of a series of subsets these agents could search the subsets to devise a solution. The agents would have common access to all data and would communicate and make adjustments among themselves until the solution is reached. In terms of IS design methodology, our information system would run in a way that processes which constitute information system subsystems are first isolated from the data process/class matrix.



ıation system

TABLE 1:

PSEUDO-CODE FOR MULTI-AGENT SYSTEM

Agent 1	Agent 2	Agent 3,4
Input: process/class matrix **Output**: set of subsystems 1. Execute minimum k-way cut on process/class matrix and find IS subsystems	**Input**: agent 1 output **Output**: linear order of execution 1. Find starting order matrix through data class delivery rule 2. Find final solution through evolutionary improvement	Still in early stages of development.

Take note that agent 1 and 2 can run simultaneously if agent 2 takes process/class matrix as its input. This can reduce execution time of the multi-agent system considerably.

A process is a series of activities that are relevant for the business system while a data class refers to any formalized data set relevant for the business system that circulates among IS processes/subsystems. In our approach, on the basis of the isolated information system

doi:10.4304/jsw.8.7.1660-1665

subsystems a linear order of executing IS subsystems would be developed with a minimum number of feedbacks to ensure that the given IS is more appropriately implemented and tested. The next step in the automation would be automatic database normalization with a minimum possible redundancy. The last step in the automation would be the development of a certain descriptive language by means of which we would automatically generate source code (at least global code structure with already implemented data retrieval and display). We demonstrated and described the first part of creating IS subsystems from the process/class matrix in [1],[2]. We also performed and published the second step in [3],[4]. Automatic database normalization and grammar development are still in progress. The system would thus be divided into at least four agents, each of which would be assigned particular operations. The common output for all the agents would constitute the final solution that would represent a successfully automatedly designed and partially implemented IS.

So far in our research the first two agents have almost entirely been developed, in which a lot of artificial intelligence algorithms was implemented including k-way cuts, as described in [1] [2]. An evolutionary approach to solving a second agent problem was used in [4] [3]. Furthermore, we adapted a lot of algorithms or re implemented new ones that are more suitable for our specific problem.

II. Related Work

At the beginning of our research we proposed an algorithm that solves the problem of determining the IS subsystems execution and implementation order, this execution and development order is done by second agent [4]. As suggested in [4], the execution and implementation order should not be performed arbitrarily or by taking into account only business rules, which is common practice. Instead, data classes that are exchanged between IS subsystems should be observed. The reason why we opted for such an approach are various problems that arise in IS implementation, execution and testing. Therefore we assumed, and later demonstrated, that the subsystem that is least dependent on the rest of the IS should be implemented first, since in that case that subsystem will have a major part that can be implemented and tested. Moreover, since in that case it will depend on a small part of the remaining IS it will be easier to fully implement it [4]. Naturally, exceptions will occur in which the system is so complex that the starting point of the execution is of no consequence. However, it is unlikely that this will continuously be the case considering that most systems are small-scale. Since, at the beginning of our research, we found that a number of the algorithms exist that are somewhat identical though not entirely compatible, we decided to develop a new algorithm that will be elaborated, analyzed, implemented and tested in detail. The algorithms that were most similar to problem of linear ordering of IS subsystems were topological sorting [5]-[8] and the Hamiltonian path [9]-[15]. Topological sorting is the algorithm first described by Kahn [8]. This algorithm is very efficient and can solve problem in a very short time. Unfortunately, since topological sorting is only applicable to a DAG we were not able to use that algorithm in our problem. The Hamiltonian path was first described by Hamilton [13], [14]. A Hamiltonian path refers to a path that visits each node exactly once, whereas a Hamiltonian cycle, as suggested by its name, also involves a cycle [13]-[15]. It was not possible to apply the Hamiltonian cycle to our problem considering that in our case adjacent nodes in a linear order do not need to be connected, which is not in accordance with the Hamiltonian cycle definition. Our main concern was to obtain a global minimum number of feedbacks, while the connection with adjacent nodes is irrelevant. Since during literature review we did not find an algorithm that would be directly applicable to our problem we decided to develop a new algorithm that would be appropriate for its resolution. The description of the developed algorithm is provided here. Owing to problem complexity, which makes devising any kind of order difficult, we decided to divide the algorithm into two parts. In the first part heuristics would be used to determine the order that is assumed to be very close to the final solution, whereas the second part would actually enable for the final solution to be obtained. In other words, it would iteratively find increasingly better solutions until the final solution, i.e., a minimum number of feedbacks, is achieved. In the first part, the LPT rule [16], [17] is used to distribute processes to be executed in a way that the process with the longest processing time comes first. The algorithm aims to find the minimum time for running a given set of processes by running them in shortest job last order so as to avoid process accumulation that would prolong their execution. This algorithm is analogously applicable to our problem regarding that in our case the execution time is represented by feedbacks that are implied in the adjacency matrix. The LTP rule has proved to generate near-optimal solutions in general.

A. Multi-agent System Research so far: Agent 2

The algorithm first calculates the number of entry and exit connections in accordance with the formulas in [4].

$$\sum_{j=1}^{n} D(E)_{\bar{\imath}j} \quad (1)$$

$$\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{D(E)}_{\mathrm{i\bar{j}}} \quad (2)$$

Based on the calculated values of entry and exit connections in the adjacency matrix we determine the starting IS subsystems execution order [4] using the aforementioned rule, i.e., greedy algorithm [18].

After that, subsystems are first ordered in accordance with

$$\max[\mathrm{v_i} \overset{\mathrm{out}}{\longrightarrow} \mathrm{D(E)_i}] \quad (3)$$

which means that the subsystems with a maximum number of exit connections that are considered to service a large number of subsystems, thus enabling fewer feedbacks in the entire order, should be scheduled first [4]. In case that the maximum number of exit connections is not unique entry, connections are considered as follows [4]:

$$\min\left[\mathrm{v_j} \overset{\text{in}}{\rightarrow} \mathrm{D(E)_j}\right]. \quad (4)$$

Subsystems with the maximum number of entry connections are pushed to the end of the cycle since they are data class consumers that, if found in the initial position, will create a large number of feedbacks in the final solution. Therefore, in case of identical subsystems the subsystem that has a minimum number of entry connections will be selected and scheduled at the end of the predefined linear order [4]. If the maximum number of entry connections is equal to the maximum number of exit connections, either can be selected and scheduled at the end of the existing order, considering that no criteria by which we would determine subsystems priority exists. If the adjacency matrix is expressed as [4]

$$\begin{array}{lll} 1 \rightarrow 3[2] & 1 \rightarrow 2[2] & \\ 2 \rightarrow 3[8] & 2 \rightarrow 1[1] & \\ 3 \rightarrow 1[1] & 3 \rightarrow 2[3] & 3 \rightarrow 4[2], \\ 4 \rightarrow 2[3] & 4 \rightarrow 5[1] & \\ 5 \rightarrow 2[1] & & \end{array} \quad (5)$$

in accordance with formulas (1) and (2) we get

TABLE 2.

IN / OUT DEGREE OF SUBSYSTEMS REPRESENTED BY (1) [4]

Subsystem	In-Degree	Out-Degree
1	2	4
2	9	9
3	10	6
4	2	4
5	1	1

Next, as described in [4], on the basis of feedbacks we need to determine the initial order that will be further improved later. After calculating feedbacks in accordance with (3) and (4) and transferring results via subsets, as described in [4], we get the initial order of subsystems that will be improved until the final solution is reached. The initial matrix of subsystems upon which further improvements will be performed is as follows:

TABLE 3.

INITIAL ORDER MATRIX (IOM) [4]

Subsystem	In-Degree	Out-Degree
2	9	9
3	10	6
1	2	4
4	2	4
5	1	1

TABLE 4.

PSEUDO-CODE FOR IOM ALGORITHM [4]

IOM Algorithm

Input: adjacency matrix
Output: IOM matrix

1. Find maximal $\mathbf{D(E)_i}$ for $\mathbf{v_i}$ in the set of subsystems from the adjacency matrix.
2. When a subsystem with $\mathbf{max}[\boldsymbol{v_i} \overset{\boldsymbol{out}}{\longrightarrow} \boldsymbol{D(E)_i}]$ is found, that subsystem is eliminated from the set IS and added at the end of the sorted set IS´. The set IS needs to be searched as long as IS≠∅.
3. If $\mathbf{max}[\boldsymbol{v_i} \overset{\boldsymbol{out}}{\longrightarrow} \boldsymbol{D(E)_i}]$ is not unique move subsystems into IS´´ and calculate in degree.
4. Find $\mathbf{min}\left[\boldsymbol{v_j} \overset{\boldsymbol{in}}{\rightarrow} \boldsymbol{D(E)_j}\right] \boldsymbol{D(E)_j}$ for $\boldsymbol{v_j}$ in the set IS´´.
5. Eliminate found subsystem from the set IS.
6. Revert set IS´´ to ∅. The set IS needs to be searched as long as IS≠∅.
7. If $\mathbf{min}\left[\boldsymbol{v_j} \overset{\boldsymbol{in}}{\rightarrow} \boldsymbol{D(E)_j}\right]$ is not unique select either subsystem from a set of equals, eliminate it from the set IS and add it at the end of the sorted set IS´.
8. The set IS´´ is reverted to ∅. The set IS needs to be searched as long as IS≠∅.

Once the initial order has been obtained, we need to consider whether it represents the final solution. Common logic suggests that it does not, taking into account the used algorithms on the one hand and the widely-established theoretical foundations on the other. This solution therefore needs to be improved in terms of the feedback/feedforward ratio, as described in [4]. Further improvements are constantly made by examining the existing order in accordance with the formula as stated in [4]

$$\max\left(\left\{\sum_{\mathrm{k=1}}^{\mathrm{p-1}}\left[\mathrm{ISs_p(Bc_{p-k})} - \mathrm{ISs_{p-k}(Fc_p)}\right]\right\} \leq 0\right) \quad (6)$$

As described in [4], if we apply formula (6) to the adjacency matrix, i.e., the initial order in table 2, where the initial order is expressed as

$$\mathrm{SO} = \{2, 3, 1, 4, 5\}, \quad (7)$$

the final solution

$$\mathrm{SO} = \{1, 4, 5, 2, 3\} \quad (8)$$

is obtained in which further improvements are not possible so the final solution is the one with the best ratio of feedbacks and sequential connections, as illustrated in figure 3.

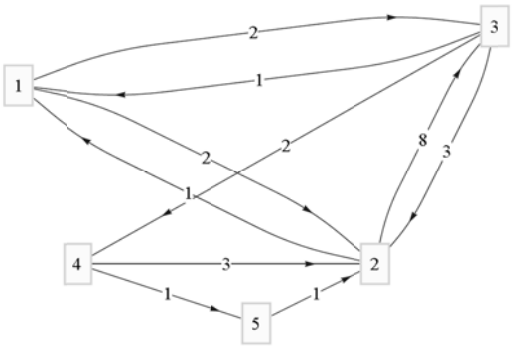


Figure 2. Best linear order for (5) [4]

Once the algorithm that is briefly described above (for details see [4]) has been developed we proceeded with algorithm implementation and its analysis published in [3]. Considering that the algorithm is divided into four parts it was necessary to implement formulas (1), (2), (3), (4) and (6). According to [3] an algorithm is divided into:

1. method for finding in-degree, (1),
2. method for finding out-degree, (2),
3. method for finding starting order, (3) and (4),
4. method for improving starting order and incrementally finding final solution (6).

After the implementation we performed complexity calculation. The complexity of the first function is $O(n^2)$, which, considering the used data structure (a two-dimensional array) selected for its performance, represents appropriate complexity for the simple reason that it is proportional to the size of the data structure that holds the data. In other words, we can consider this function to be very efficient [3]. The complexity of the other two functions is also $O(n^2)$ so the same interpretation can also apply to them, while the complexity of the fourth function is higher than that we would find satisfactory [3], amounting to $O(n^3)$. The complexity of the fourth function is not aligned with the size of the data structure that holds the data since additional calculations are necessary for the final solution to be achieved, which itself implies higher complexity [3].

As suggested in the previous paper [3], $O(n^3)$ represents satisfactory complexity with regards to the number of IS subsystems that can occur in a real-world example. However, from a purely theoretical point of view or in terms of the potential of the implementation of the algorithm on a problem with a large number of nodes to be resolved, the question remains whether even such complexity would be satisfactory. We therefore decided to conduct empirical analysis of the algorithm to determine what its limitations are, that is, to identify cases in which the algorithm runs satisfactorily as well as those in which that is not the case.

III. EMPIRICAL ANALYSIS

Before presenting the results of the empirical analysis, we will describe the procedure in which the analysis was conducted and the data that the algorithm was tested upon. With regards to the complexity of functions that we discussed in the previous section, we will only test the function that obtains the final solution, since that is the function with highest complexity ($O(n^3)$) that also represents the exact place in the algorithm where a bottleneck will occur concerning execution time. In addition, it is the only function whose complexity is not aligned with the size of the data structure that holds the data. In terms of the data to be tested, we will use an accidental number generator to generate the adjacency matrix as the entry parameter for our algorithm. Since the accidental number quality, sufficient randomization, is of no consequence in our case, it is not necessary to use more complex algorithms for accidental number generation. We will therefore use the default class for accidental number generation in C#. The data will be randomized so that we determine the number of subsystems that the agent, i.e., the algorithm, will run on, while the function for accidental number generation will help us to fill large matrices. As a matter of fact, during testing it is not necessary and would not be advisable to fully randomize the algorithm since with this type of problems we already know in which cases malfunction will occur. We will thus conduct testing in a way that we will ourselves regulate the parameter of the number of subsystems between which connections will be generated so that the most accurate possible data are obtained. Entering the number of subsystems (nodes) manually will enable higher flexibility during testing. Conducting worst case testing would be recommendable although that would require that the problem of the maximum number of feedbacks in an IS, which in ideal case is as complex as the present problem, is resolved first. If this testing approach should prove unsatisfactory, we would obtain results that are highly unexpected, so we would proceed with solving the former problem as well. The results of the empirical analysis are provided here. We will generate connections for each number of subsystems. For each case we will measure the time needed for the agent to obtain the final solution. In addition, for each case we will change the maximum random number that a random function can generate to establish the dependence between the number of subsystems and the number of connections. Next we will analyze the results and create tables to show solution times for each case with regards to the total number of connections for a particular number of subsystems. Finally, for each pair (number of subsystems-number of connections) the mean value will be calculated so as to obtain the measure of central tendency of that case regarding the generated data.

First we included five subsystems for which connections were generated. The results are as follows:

TABLE 5.

TESTING AGENT ON 5 NODES

5 Subsystems	Time for number of connections (number of connections = Subsystems x n, n[2, 3, 4]) / sec		
	10 connections	15 connections	20 connections
	0,0000145	0,0000489	0,0000339

	0,0000241	0,0000958	0,0000921
	0,0000693	0,0000724	0,0000857
	0,0000164	0,0000844	0,0000795
	0,0000175	0,0000283	0,0000479
Mean	0,0000**284**	0,0000**660**	0,0000**678**

As table 4 shows, in this case everything will work flawlessly, which was to be assumed from [4]. There is a slight difference in the increase of the number of connections, but it is negligible. However, if a difference amounting to a unit of time can be noticed with such short times, it is to be expected that for a greater number of subsystems that difference would be even more noticeable. That difference cannot be attributed to running the entire application since only the execution time of the code that performs the calculations is measured. After the first test we decided to run another test on 20 subsystems taking into account that this is the maximum number of subsystems that a company of an average or above-the-average size should comprise. The results are as follows:

TABLE 6.

TESTING AGENT ON 20 NODES

20 Subsystems	Time for number of connections (number of connections = Subsystems x n, n[2, 3, 4]) / sec		
	40 connections	60 connections	80 connections
	0,0003097	0,0004321	0,0004577
	0,0004176	0,0003063	0,0003323
	0,000331	0,0003268	0,000581
	0,0004415	0,0003221	0,0002227
	0,0004168	0,0003182	0,0003425
Mean	0,000**3833**	0,000**3411**	0,000**3872**

If we observe the second test with 20 subsystems, it is interesting that there are hardly any differences between results. However, although the times, when compared to the previous test, are short, a considerable difference in the increase of execution time is still noticeable. It is evident that this difference arose from the selection of the initial order for which the final solution aims to be obtained. In other words, the algorithm performed a lot fewer replacements in the first test than in the second one, which was to be expected regarding the increase in the number of subsystems and the number of connections. Hence, since the agent, i.e., the part of the algorithm referred to as subsystem in the introduction to system description, works satisfactorily for 20 subsystems, which is more than sufficient for implementation purposes in IS design and implementation, no further code optimization is required. Consequently, the problem concerning its practical application that we referred to in [3] and [4], prior to the empirical analysis which showed that concerning problem complexity everything had properly functioned, does not need to be resolved either.

After these two tests we decided to investigate the extreme limitations of the algorithm regarding the increase in the number of subsystems and the total number of connections between subsystems, and, consequently, permutations.

First considerable delays started to occur with 250 subsystems and 500 connections, where the time needed for calculating the final solution started to reach 1.7 seconds. With 500 subsystems and 100 connections between subsystems the time needed for calculating the final solution rose to 7 seconds and, in some instances, to even 10 seconds. With a further increase in the total number of connections between subsystems and, especially, in the number of subsystems, the number of permutations needed for calculating the final solution is drastically increased, which also implies an increase in execution time. Therefore, regardless of reasonable complexity $O(n^3)$, problem complexity does not allow for the algorithm to be executed within one second for 200 subsystems and more. Although the execution time that amounts to, for instance, several minutes or half an hour would be satisfactory for problems where an instant solution is not imperative since with 500 subsystems we could easily design information system of any kind. With problems that have a huge number of subsystems and work in real time or near-real time this algorithm would no longer be usable. Alternatively, it would either have to be further optimized or the aforementioned delays would have to be acceptable.

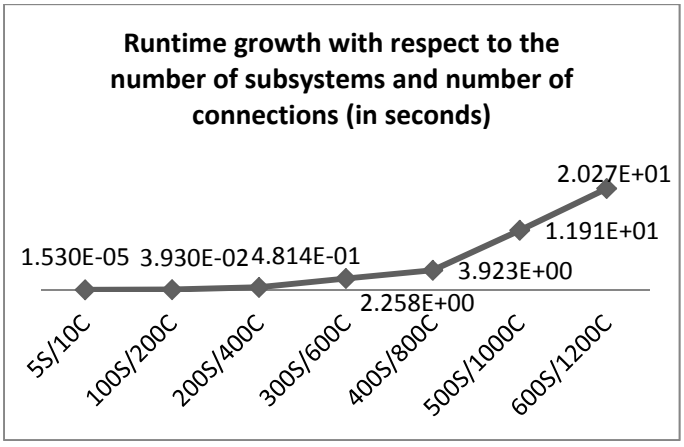


Figure 3. Runtime order of growth for tested function

The graph in figure 3 shows a comparative test of algorithm execution with regards to the increase in the number of subsystems and connections between them. Since the complexity of the third function is a third-order polynomial, the graph also represents the cubic function growth. Unfortunately, with a large number of permutations such complexity is not beneficial.

Furthermore, as evident in the previous results, the number of connections between subsystems is itself of no consequence since with a small number of permutations the difference is hardly noticeable. Likewise, with a large number of permutations or subsystems the number of permutations is too large for the difference to be

noticeable since, in general, the total number of connections between subsystems that is less the number of subsystems is not to be expected. This is explained by the fact that those connections, i.e., data classes [4] need to service particular IS subsystems so the IS could work properly. Otherwise the question would arise of whether that whole, if it is so poorly interconnected, can be referred to as a system.

IV. CONCLUSION

The goal of this part of research was to examine the functioning of an algorithm we previously developed [3],[4] in real conditions beyond theoretical considerations. Having conducted the empirical analysis of the agent for automatic determination of IS subsystems order execution, we can conclude that the algorithm proved to be suitable for application during IS design and implementation. Furthermore, if the number of subsystems to be processed is large and it is necessary to run a large number of permutations without the real time operation requirement, the algorithm is applicable and works satisfactorily. The only issue concerning the application of this algorithm is related to using it in conditions in which an almost simultaneous response is expected and the number of nodes it should be run on is huge. Although it is true that the algorithm had not been developed for such conditions, from the purely scientific perspective it is possible to apply the algorithm to that type of a problem as well.

As mentioned in our previous research, we assumed that such a situation would occur and here reaffirm our conclusion that an approximation algorithm could be developed by means of topological sorting and resolution of cycles that would be much less complex and would work properly. Naturally, when such an algorithm is developed it will need to be compared with the one presented in this paper. Moreover, considering our research in machine learning [19],[20], automatic or semiautomatic generation of the process/data class matrix is possible, although it may be too early to discuss that option. Also, it is possible that approximation algorithm could be developed with DFS [18],[21],[22],[23] as well, but, this notion would require more thorough research.

ACKNOWLEDGMENT

This research was conducted within the project “Automation of Procedures in Information Systems Design” financed by the Croatian Ministry of Science, Education and Sports.

REFERENCES

[1] A. Lovrenčić, An efficient algorithm for information system decomposition, Journal of Information and Organizational Sciences, Vol. 22, n. 2, pp. 137-151, 1998.
[2] A. Lovrenčić, The problem of optimization of the process of decomposition of an information system, Journal of Information and Organizational Sciences, Vol. 1, n. 22, pp. 27-43, 1997.
[3] Robert Kudelić, Alen Lovrenčić, Mladen Konecki, Information system subsystems execution and development order algorithm implementation and analysis, International Journal of Computer Science Issues, Vol. 9, Issue 2, No 3, March 2012.
[4] R. Kudelić, A. Lovrenčić, Automatic determination of information system subsystems execution and development order, IRECOS (2011).
[5] Huang Wei J., Cai Li Gang, Hu Yu Jing, Wang Xue L., Ling Ling, Process planning optimization based on genetic algorithm and topological sort algorithm for digraph, Jisuanji Jicheng Zhizao Xitong/Computer Integrated Manufacturing Systems, Volume 15, n. 9, pp. 1770-1778, 2009.
[6] Moon Chiung K., Yun Youngsu S., Leem Choon Seong, Evolutionary algorithm based on topological sort for precedence constrained sequencing, IEEE Congress on Evolutionary Computation, 2007, pp. 1325-1332, 2008.
[7] Li YL. Zhang JH. Li CA, NOTE ON SOME TOPOLOGICAL PROPERTIES OF SETS IN INFORMATION SYSTEMS, Kybernetes, Volume 27, 1998.
[8] Kahn, A. B. (1962), "Topological sorting of large networks", Communications of the ACM 5 (11): 558–562.
[9] Lijiang Zhao, A matrix solution to Hamiltonian Path of any graph, Proceedings - 2010 International Conference on Intelligent Computing and Cognitive Informatics, pp. 440-442, 2010.
[10] Feng JF., Giesen HE., Guo YB., Gutin G., Jensen T., Rafiey A., Characterization of edge-colored complete graphs with properly colored Hamilton paths, Journal of Graph Theory, Volume 53, pp. 333-346, 2006.
[11] Dyer M., Frieze A., Jerrum M., APPROXIMATELY COUNTING HAMILTON PATHS AND CYCLES IN DENSE GRAPHS, SIAM Journal on Computing, Volume 27, pp. 1262-1272, 1998.
[12] Michael R. Garey, David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (W.H. Freeman, First Edition 1979).
[13] Hamilton, William Rowan, "Memorandum respecting a new system of roots of unity". Philosophical Magazine, 12 1856.
[14] Hamilton, William Rowan, "Account of the Icosian Calculus". Proceedings of the Royal Irish Academy, 6 1858.
[15] Ore, O "A Note on Hamiltonian Circuits." American Mathematical Monthly 67, 55, 1960.
[16] Ellis Horowitz, Sartaj Sahni, Fundamentals of computer algorithms, Computer Science Press, 1978.
[17] Graham, Bounds on Multiprocessing Timing Anomalies. SIAMJournal on Applied Mathematics 17, 416-429, 1969.
[18] Cormen, Leiserson, and Rivest, Introduction to Algorithms, 1990.
[19] Kudelić, R, Konecki M, Lovrenčić A. Mind Map Generator Software Model with Text Mining Algorithm. Proceedings of the 33rd International Conference on Information Technology Interfaces, 2011, p. 487.
[20] Kudelić, R, Maleković, M, Lovrenčić A. Mind Map Generator Software. International Conference on Computer Science and Automation Engineering, accepted for publication, 2012.
[21] Charles P. Tremaux, French engineer of the telegraph, 1859–1882.
[22] Even, Shimon, Graph Algorithms (2nd ed.), Cambridge University Press, pp. 46–48, 2011.
[23] Sedgewick, Robert, Algorithms in C++: Graph Algorithms (3rd ed.), Pearson Education, 2002.