Information system subsystems execution and development order algorithm implementation and analysis

Robert Kudelić[1], Alen Lovrenčić[2] and Mladen Konecki[3]

[1] Faculty of Organization and Computer Science, University of Zagreb, Varaždin, 42000, Republic of Croatia

[2] Faculty of Organization and Computer Science, University of Zagreb, Varaždin, 42000, Republic of Croatia

[3] Faculty of Organization and Computer Science, University of Zagreb, Varaždin, 42000, Republic of Croatia

Abstract

In our previous research we have constructed theoretical foundations for automated approach that can determine information system subsystems execution and development order according to data class interactions. In this paper, we will, from those theoretical foundations, develop C# algorithm through which we can see its real time behavior and calculate complexity. Finally, after algorithm analysis we will conclude with our plans for further work.

Keywords: *Information System, Subsystem, Development, Execution, Order, Algorithm, Analysis.*

1. Introduction

Determining right development and execution order of information system subsystems, according to its data class creation and consumption [1], is a problem that has its roots in graph theory. Therefore, when we tried to solve this problem first thing we did was to look for the solution in graph theory. After some time we have found some solutions that are similar to our problem. Topological sort, as it can be seen in [2, 3, 4, 5], is an algorithm for directed acyclic graphs (DAG), where each node is sorted into a linear order observing the nodes precedence constraint [1]. However, data in an information system often circulates through several subsystems, going back and forth several times and since topological sort only works on DAG and cannot be applied to cycles, we could not implement it in our solution [1]. Also, a well-known problem in graph theory is the Hamiltonian path problem, which seeks a path that visits each node exactly once [1], as it can be seen in [6, 7, 8, 9]. Nevertheless, since this path or cycle, in case of the Hamiltonian cycle, does not eliminate cycles through linear ordering of nodes but rather seeks a path that visits each node only once, searching for this path could not be used in our case either [1]. This can be explained by the fact that graph that represents the problem we wanted to solve typically does not contain those paths, since in our linear order adjacent nodes do not have to be connected [1]. We could not find appropriate existing solution, therefore we have developed, as it can be seen in [1], new algorithm that analyzes a directed cyclic graph (DCG), which represents information system subsystems data class interaction, and provides us with the linear order of graph nodes (information system subsystems) with a minimal number of backward connections. In this paper we will implement aforementioned algorithm in C# programming language, thus we will be able to analyze developed algorithm and see its real time behavior in one of the most used programming languages today.

2. Basic notions

In this section we will give basic notions that are necessary to understand algorithm implementation that has been done in this paper. Firstly, we will briefly describe problem solved in [1] so it is clear what we are talking about. Secondly, we will lay out results, from our previous research that can be found in [1], that are needed for algorithm implementation. Our problem is defined with directed cyclic graph G = (V, E), where E is the edge between information system subsystems that represents data classes circulating between those subsystems and V are vertices that represent the actual IS subsystems [1] that are created according to the algorithm for information system decomposition proposed by Lovrenčić in [10, 11]. Also, DCG has a degree on each edge where $D(E) \geq 1$ [1]. The problem arises when we want to know which of the information system subsystems should be implemented and executed first. For example, let us assume that linear order (LO) of G, as it can be seen in [1],

$$\mathrm{LO(G)} := \{\mathrm{ISs_1}, \dots, \mathrm{ISs_{i-1}}, \mathrm{ISs_i}\}, \quad (1)$$

is in our case,

$$LO\{ISs_1, ISs_2, ISs_3, ISs_4, ISs_5\}. \quad (2)$$

Also, relation for our graph G [1] looks as follows,

$$\begin{matrix} 1 \to 3[2] & 1 \to 2[2] & \\ 2 \to 3[8] & 2 \to 1[1] & \\ 3 \to 1[1] & 3 \to 2[3] & 3 \to 4[2]. \\ 4 \to 2[3] & 4 \to 5[1] & \\ 5 \to 2[1] & & \end{matrix} \quad (3)$$

Now, as it can be seen from Eq. (2) and (3). ISs_2 is implemented before ISs_3, but, ISs_2 needs data from ISs_3 to function properly [1]. This means that ISs_2 needs to work with data that has not been created yet and since ISs_3 creates more data for ISs_2, ISs_3 should be implemented and executed first [1]. From previous example we can clearly see that information system subsystems can't be implemented and tested adequately if those same subsystems do not poses data they need to work with [1]. Also, if subsystems would execute inadequately, like we have previously described, we would have unnecessary waiting for data to be processed which is not at all desirable. Therefore, we have developed algorithm, as it can be seen in [1], which solves this problem. Algorithm is divided into two phases [1]:

- determination of starting linear order,
- starting linear order improvement.

In the following section we will quickly present both phases, for full description see [1]. Determination of starting linear order is basically trying to, through heuristics (greedy algorithm) [5], guess a solution that is as close as possible to final linear order. Firstly, it brings forward, in LO, all subsystems that are creating large amounts (maximum) of data classes,

$$\max\left[v_i \overset{out}{\longrightarrow} D(E)_i\right]. \quad (4)$$

Secondly, if we encounter equal number of data classes we are then taking into consideration data class consumption and are pushing back subsystems that are consuming large amounts of data classes,

$$\min\left[v_j \overset{in}{\rightarrow} D(E)_j\right]. \quad (5)$$

Finally, if both of those are equal then it is of no consequence which subsystem will take precedence in linear order of development and execution. Starting linear order improvement phase is gradually correcting starting linear order until we can't make any more improvements. Improvements are made by constantly applying the following rule,

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

for which theoretical foundations can be found in [1].

Simply put, this rule constantly searches previously found linear order for a permutation which will give us less backward connections (BC). The first time(first pass) this rule is not valid means that we have found our final solution, since we cannot make any more adjustment on linear order that would give less BC. See [1] for calculation example and detailed explanation of theoretical foundations mentioned above. Now, when we know how this algorithm works in theory we will implement it in practice so we can analyze it and ascertain its real time behavior and complexity.

3. Algorithm implementation and analysis

In this section we will present implemented C# algorithm. Algorithm is implemented in C#, for reasons described in [1] conclusion and partly in this papers introduction. We will not present entire class that is responsible for making calculations since there is no point in presenting instantiations, data manipulation, temporary variables, etc. outside main algorithm. Therefore, we will present only core algorithm which is relevant for our analysis. First we calculate out-degree for subsystems in graph G = (V, E) given by Eq. (3) according to,

$$\sum_{j=1}^{n} D(E)_{ij}, \quad (7)$$

as described in [1].

```
private void rac_OutD()
{
     int suma = 0;
     for (int i = 0; i < DataGridView.RowCount; i++)
     {
     suma = 0;
     tMatrSus1[i,0]=DataGridView.Rows[i].HeaderCell.Valu
e;
     for (int j = 0; j < DataGridView.RowCount; j++)
             {
     suma+=Convert.ToInt32(DataGridView.Rows[i].Cells[j]
.Value);
     }
     tMatrSus1[i, 2] = suma;
     }
}
```

Then we calculate in-degree for subsystems in graph G = (V, E) given by Eq. (3) according to,

$$\sum_{i=1}^{n} D(E)_{ij}, \quad (8)$$

as described in [1].

```
private void rac_InD()
{
     int suma = 0;
     for (int i = 0; i < DataGridView.ColumnCount; i++)
     {
     suma = 0;
     for (int j = 0; j < DataGridView.RowCount; j++)
     {
     suma+=Convert.ToInt32(DataGridView.Rows[j].Cells[i]
.Value);
     }
     tMatrSus1[i, 1] = suma;
     }
}
```

Now, with out-degree and in-degree we can calculate starting order (SO) as described earlier by Eq. (4) and (5), for more detail see [1].

```
private void poc_Red()
{
     tMatrSus2 = tMatrSus1;
     temp = Convert.ToInt32(tMatrSus2[0, 2]);
     int indeks_maks = 0;
     int br = DataGridView.RowCount - 1;
     int temp2 = Convert.ToInt32(tMatrSus2[0, 1]);
     SOM = new object[DataGridView.RowCount, 3];
     int br2 = 0;
     do
     {
     for (int i = 0; i < DataGridView.RowCount; i++)
     {
```

```
     if (temp < Convert.ToInt32(tMatrSus2[i, 2]) &
tMatrSus2[i, 0] != null)
     {
     temp = Convert.ToInt32(tMatrSus2[i, 2]);
     indeks_maks = i;
     temp2=Convert.ToInt32(tMatrSus2[indeks_maks, 1]);
     }
     }
     for (int i = 0; i < DataGridView.RowCount; i++)
     {
     if (temp == Convert.ToInt32(tMatrSus2[i, 2]) &
tMatrSus2[i, 0] != null)
     {
     if (temp2 > Convert.ToInt32(tMatrSus2[i, 1]))
     {
     indeks_maks = i;
     }
     }
     }
     SOM[br2, 0] = tMatrSus2[indeks_maks, 0];
     SOM[br2, 1] = tMatrSus2[indeks_maks, 1];
     SOM[br2++, 2] = tMatrSus2[indeks_maks, 2];
     tMatrSus2[indeks_maks, 0] = null;
     for (int i = 0; i < DataGridView.RowCount; i++)
     {
     if (tMatrSus2[i, 0] != null)
     {
     temp = Convert.ToInt32(tMatrSus2[i, 2]);
     indeks_maks = i;
     temp2 = Convert.ToInt32(tMatrSus2[indeks_maks, 1]);
     }
     }
     br--;
     } while (br >= 0);
}
```

Finally, with starting order we can incrementally calculate final solution according to SO, Eq. (3) and (6).

```
private void kon_rjes()
{
     object[,] tRazmjSus = new
     object[DataGridView.RowCount - 1, 2];
     object[,] tSum = new object[DataGridView.RowCount -
1, 2];
     bool jos = false;
     int indeks1 = 0, indeks2 = 0, br4 = 0, suma = 0,
     br5 = 0, maxV = 0;
     object t1, t2, t3;
     do
     {
     jos = false;
     for (int i = 1; i < DataGridView.RowCount; i++)
     {
     for (int k = 0; k < DataGridView.RowCount; k++)
     {
     if (SOM[i, 0] == pocMatrSus[k, 0])
     {
     indeks1 = k;
     break;
     }
     }
     for (int j = 0; j < i; j++)
     {
     for (int h = 0; h < DataGridView.RowCount; h++)
     {
     if (SOM[j, 0] == pocMatrSus[h, 0])
     {
     indeks2 = h + 1;
     tRazmjSus[j, 0] = SOM[j, 0];
     tRazmjSus[j,1]=Convert.ToInt32(pocMatrSus[indeks1,i
ndeks2]) - Convert.ToInt32(pocMatrSus[indeks2 - 1,
indeks1 + 1]);
     br4++;
     break;
     }
     }
     }
     for (int q = br4 - 1; q >= 0; q--)
     {
     suma += Convert.ToInt32(tRazmjSus[q, 1]);
     tSum[q, 0] = tRazmjSus[q, 0];
     tSum[q, 1] = suma;
     }
     for (int h = br4 - 1; h >= 0; h--)
     {
     if (Convert.ToInt32(tSum[h, 1]) > maxV)
     {
     maxV = Convert.ToInt32(tSum[h, 1]);
     br5 = h;
     }
     }
     if (maxV > 0)
     {
     t1 = SOM[i, 0];
     t2 = SOM[i, 1];
     t3 = SOM[i, 2];
     for (int n = i - 1; n >= br5; n--)
     {
     SOM[n + 1, 0] = SOM[n, 0];
     SOM[n + 1, 1] = SOM[n, 1];
     SOM[n + 1, 2] = SOM[n, 2];
     }
     SOM[br5, 0] = t1;
     SOM[br5, 1] = t2;
     SOM[br5, 2] = t3;
     jos = true;
     }
     br4 = 0;
     br5 = 0;
     suma = 0;
     indeks1 = 0;
     indeks2 = 0;
     suma = 0;
     maxV = 0;
     t1 = null;
     t2 = null;
     t3 = null;
     Array.Clear(tRazmjSus, 0, DataGridView.RowCount -
1);
     Array.Clear(tSum, 0, DataGridView.RowCount - 1);
     }
     } while (jos);
}
```

As it can be seen algorithm is divided into four major parts:

- method for finding in-degree,
- method for finding out-degree,
- method for finding starting order,
- method for improving starting order and incrementally finding final solution.

Methods for in/out-degree calculation are calculating in and out degree for every subsystem in a graph G = (V, E) according to Eq. (3), (7) and (8) respectively. When in and out degree is calculated we are executing method for starting order calculation. Method for calculating starting order takes previously calculated in/out degree and, according to Eq. (4) and (5) described in previous chapter (for more details see [1]), calculates starting linear order. Now, when we have starting linear order we are applying Eq. (6) on SO and Eq. (3) and gradually improving linear order until we can't make any more improvements. When method which seeks final solution executes and does not find candidates for position change we know that this is the best solution. We can see from this description as well as from source code that method for finding final solution needs to execute one more time when final solution is found since we do not know in advance that this is the end (like bubble sort [12]). When we take a look at developed algorithm data structures we can see that arrays are used

exclusively. There is a reason for that. If you look closely at the problem of finding development and execution order and graph G = (V, E) that visually describes it, first data structure that comes to mind is a list. Nevertheless, as it can be seen in [10, 11] and other practical examples, dynamic memory allocation and constant list manipulation and memory allocation becomes rather slow process when you constantly need to allocate extremely large amount of list elements. Since, we know exactly how much memory is necessary for our calculations and data manipulation and since these days there is no shortage of memory we can immediately allocate memory we need. Therefore, we have ensured, at least from this point of view, that developed algorithm will run faster. In the following section we will comment on calculated algorithm complexity. Complexity for the function rac_OutD() is $O(n^2)$. If we take into account that complexity of this function is proportional to the size of data structure that holds data for the problem, this function is very efficient. The same can be said for the second, rac_InD(), and the third, poc_Red(), function since complexity for these functions is also $O(n^2)$. Unfortunately, the fourth function, kon_rjes(), which calculates final solution does not follow previous functions regarding complexity. This function complexity is $O(n^3)$. However, when we take into account difficulty and seriousness of the problem this was to be expected. From a practical standpoint $O(n^3)$ is a good complexity, nevertheless, from a pure theoretical standpoint it could be better. Namely, we would be satisfied if this function had worst case complexity of $O(n^2)$, like previous functions, since this function also stores its data in two-dimensional array, unfortunately this function needs to execute additional calculations in order to achieve its final state and find final solution.

4. Last word

In this paper we have implemented and analyzed algorithm for which we have previously developed theoretical foundations. When we take into consideration implemented algorithm, its complexity and difficulty of the problem we would argue that the problem we were trying to solve in information system development and implementation is solved. Nevertheless, as we mentioned in our previous research, this implemented algorithm will be empirically analyzed so we can more precisely ascertain its real time behavior. We have done some minor testing of the algorithm but this is far from sufficient. Also, there is a possibility that this algorithm can be applied to similar problems in information system development and implementation field of research. Finally, this paper has shown that, at least from theoretical standpoint, there is still room and need for further optimization. With that in mind, we will try either to further optimize current algorithm or develop approximation algorithm of a lesser complexity, as stated above preferably quadratic.

Acknowledgments

This research is funded by Croatian Ministry of Science and is a part of a larger project “Automation of Procedures in Information System Design”.

References

[1] R. Kudelić, A. Lovrenčić, Automatic determination of information system subsystems execution and development order, IRECOS (2011).
[2] 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 (2009) 1770-1778.
[3] 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.
[4] Li YL. Zhang JH. Li CA, Note on Some Topological Properties of Sets in Information Systems, Kybernetes, Volume 27 (1998).
[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms (MIT Press and McGraw-Hill, 2001).
[6] Lijiang Zhao, A matrix solution to Hamiltonian Path of any graph, Proceedings - 2010 International Conference on Intelligent Computing and Cognitive Informatics (2010), pp. 440-442.
[7] 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 (2006) 333-346.
[8] Dyer M., Frieze A., Jerrum M., Approximately Counting Hamilton Paths and Cycles in Dense Graphs, SIAM Journal on Computing, Volume 27 (1998) 1262-1272.
[9] Michael R. Garey, David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (W.H. Freeman, First Edition 1979).
[10] 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 (1997) 27-43.
[11] A. Lovrenčić, An efficient algorithm for information system decomposition, Journal of Information and Organizational Sciences, Vol. 22, n. 2 (1998) 137-151.
[12] D. Knuth, The Art of Computer Programming: Sorting and Searching, Volume 3 (Addison-Wesley, 1998).

Robert Kudelić received his masters degree in Information Science (2009) from the Faculty of Organization and Informatics, University of Zagreb where he is currently a teaching and research assistant. He is a researcher on the scientific project Automation of Procedures in Information System Design.

Alen Lovrenčić received his masters degree (1999) and his PhD (2004) from the Faculty of Organization and Informatics, University of Zagreb. He is currently associate professor at the Faculty of Organization and Informatics. He was/is a leader of several scientific projects and was/is involved with working on several journals/conferences.

Mladen Konecki received his masters degree in Information Science from the Faculty of Organization and Informatics, University of Zagreb where he is currently a teaching and research assistant. He is a researcher on the scientific project Automation of Procedures in Information System Design.