Acta Universitatis Sapientiae

Informatica

Volume 8, Number 1, 2016

Sapientia Hungarian University of Transylvania
Scientia Publishing House

Contents

DE GRUYTER OPEN

Acta Univ. Sapientiae, Informatica **8,** 1 (2016) 5–15

DOI: 10.1515/ausi-2016-0001

Distance-constrained grid colouring

László ASZALÓS
Faculty of Informatics
University of Debrecen
email: aszalos.laszlo@inf.unideb.hu

Mária BAKÓ
Faculty of Economics
University of Debrecen
email: bakom@unideb.hu

Abstract. Distance-constrained colouring is a mathematical model of the frequency assignment problem. This colouring can be treated as an optimization problem so we can use the toolbar of the optimization to solve concrete problems. In this paper, we show performance of distance-constrained grid colouring for two methods which are good in map colouring.

1 Introduction

The problem of graph colouring is a very active research field nowadays, with very long history. There are different research directions. Some are interested in the theory, namely what the chromatic number of a particular type graph is, i.e. finding the minimum number of colours to colour the nodes of the graph, such that no edge connects nodes of the same colour. Others are interested in practical things, hence invent algorithms that generate colouring with minimal numbers of colours for any, given type of graphs. As the three-colouring is NP-hard problem, we cannot require any algorithm to give a quick solution for every graph.

Computing Classification System 1998: G.1.6
Mathematics Subject Classification 2010: 05C15
Key words and phrases: min-conflicts method, constraint logic programming

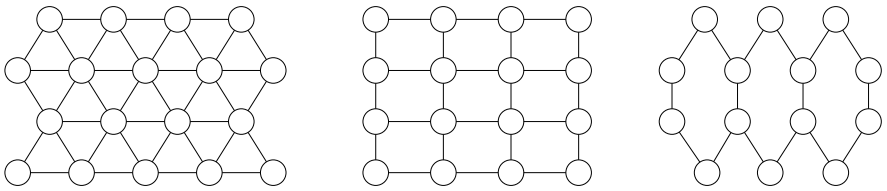

Figure 1: Triangular, square and hexagonal grids of size 4×4

If the graph is fixed, then its colouring with minimal number of colours can be treated as an optimization problem. It is not surprising that almost all of the optimization method have a variant to solve colouring problems [2].

The most elementary method is the exhaustive search for a colouring. This does not mean, that we need to try all the assignment of colours to nodes. If a partial colouring violates any constraint, there is no reason to continue, we can backtrack and start another colouring. Fifth chapter of [6] contains several heuristics to speed up this search by eliminating cases which cannot lead to solutions. This chapter contains an example where the 4-colouring of the map of USA uses around 60 steps instead of 4^{50}, by applying most of the heuristics.

The method of minimal conflicts was introduced in [4]. To solve the 4-colouring of USA map, this method needs about 60 steps too.

Both methods work well for colouring real world maps, but what about colouring other type of graphs? Previously we examined random graphs, so for us their colouring is interesting, but for the sake of reproduction and scalability we will examine very simple graphs: grids (Fig. 1). Although the degrees of the nodes are much smaller than in an ordinary random graph, we hope to get interesting results.

If the number of colours is big, then we use numbers instead of colours. Hence a k-colouring of a graph $G = \langle V, E \rangle$ mathematically is a function $c : V \to \{1, \ldots, k\}$.

The original graph colouring problem has many variants, which are actively researched. One of them is the distance-constrained colouring problem raised from the frequency assignment [3]. In this case given an n-tuple of integers, let say $\langle x_1, x_2, \ldots, x_n \rangle$ and if $d(u, v) = i$ then $|c(u) - c(v)| \geq x_i$, where $1 \leq i \leq n$, $x_j > x_{j+1}$ for $1 \leq j < n$, u and $v \in V$, and d is the graph-distance of the nodes. This distance constrain is denoted by $L_{x_1 \ldots x_n}$ in the literature. Namely at L_{21} colouring the nodes at distance 2 need to be different; and at adjacent nodes the difference of the colour codes is at least 2. Figure 2 shows L_1, L_{21} and

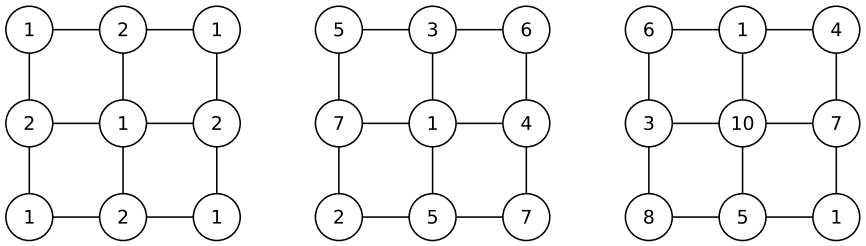

Figure 2: L_1, L_{21} and L_{321} colourings

L_{321} colourings of the 3×3 square grid. On the left, two colours were enough, because only the adjacency counts. At the centre we used seven colours, such that the L_{21} constraints would be satisfied. One could think that less colours might be enough, but no. As we shall see in the following, it is a reasonable step to colour the central node to 1. The central node is adjacent to four nodes, so all of them is 3, or more. These four nodes are second neighbours to each other, so they need to be different. Therefore we need to use at least 6 colours. The nodes in the corner are second neighbours of the central node, so none of them could be 1. As some of them are second neighbour of each others, they cannot be equal. We left to the reader to check the cases and prove, that 6 colour is not enough in this case. Finally on the right there is a 10-colouring, which satisfies L_{321} constraints.

The theoretical results about distant-constrained colouring are surveyed in [5, 1]. In this article we use practical view-point. In the next two section we present the methods we use, and next we discuss our experimental results. Finally we propose new directions.

2 Constraint satisfaction problems

Many problems can be expressed as a constraint satisfaction problem (CSP). Here given an n-tuple of variables: $\langle X_1, \ldots, X_n \rangle$, the tuple of domains of the variables $\langle D_1, \ldots, D_n \rangle$, and a set of constraints which are predicates—boolean valued function of these variables.

Our task is to select values of the variables from the corresponding domains in a such way that all the predicates become true. As the problem SAT could be transformed into CSP, the solution of CSP is NP-hard.

The simplest solution may be gained by using the backtrack method. It

does not use the information given in the problem, so it is usually slow. The 4-colouring of USA map takes more than 10^6 steps. By using variable and value ordering heuristics (minimum remaining values, degree heuristic, least constraining value, forward checking) we can dramatically reduce the number of steps [6].

Many software were built for solving CSP, some of them are stand-alone, while others are modules or libraries and enable us to include the solver in our programs. For its simplicity we have chosen the SWI-Prolog system, and its CLPFD library. Figure 3 contains the source of L_{21} constraints for the coloring of the 3×3 square grid.

Here in the first line we signal the system that we want to use the CLPFD library. Next we define a new predicate (`grid`) which is the counterpart of the function in Prolog. This predicate takes a number as an input and returns a colouring `L`, which is a list of natural numbers. The third line denotes, that the colouring is a 9 long list. The members of this list are our variables (from `X0` to `X8`). The forth line states, that these variables can have `N` different values, the natural numbers from `1` to `N`. The last line of the source starts the systematic search, which determines the values in the list `L`. The word inside the brackets in this line (`ffc`) is a search strategy. The CLPFD library has five different strategies, and this one fits our needs and aims the best. The other lines not mentioned yet contains the constraints. As we need an L_{21} colouring, in the case of adjacent variables (e.g. `X0` and `X1`) the difference of the values is at least 2, so we need to use the absolute value function. Moreover, as the variables have no value at the beginning, instead of relation $\geq$ we need to use the corresponding constraint `#>=`. In case of second neighbours we could use constraint `#\=` (not equal), but for the sake of easier generation of problems, and for the more general L_{pq} constraints we have used `#>=`, again.

Now we are ready to run this code. After loading this file we need to ask the system: `grid(6,L).` The answer arrives immediately: `false`, denoting that there is no solution using six colours. If we ask a 7-colouring with `grid(7,L).` then the answer is one solution. If the solution is too long, the system shows only its beginning. We can ask for the whole solution with a print statement: `grid(7,L), write(L).`

We believe this example shows that in case of constraint logic programming we only need to precisely state the problem, and the solver produces the solution. Moreover, it searches for solutions in a clever way—using the heuristics. If there exists at least one solution, then it determines it; and if not, then after an exhaustive search the system indicates this.

```
:- use_module(library(clpfd)).
grid(N,L) :-
  L = [X0, X1, X2, X3, X4, X5, X6, X7, X8],
  L ins 1..N,
  abs(X0-X1) #>= 2,  abs(X0-X2) #>= 1,  abs(X0-X3) #>= 2,
  abs(X0-X4) #>= 1,  abs(X0-X6) #>= 1,  abs(X1-X2) #>= 2,
  abs(X1-X3) #>= 1,  abs(X1-X4) #>= 2,  abs(X1-X5) #>= 1,
  abs(X1-X7) #>= 1,  abs(X2-X4) #>= 1,  abs(X2-X5) #>= 2,
  abs(X2-X8) #>= 1,  abs(X3-X4) #>= 2,  abs(X3-X5) #>= 1,
  abs(X3-X6) #>= 2,  abs(X3-X7) #>= 1,  abs(X4-X5) #>= 2,
  abs(X4-X6) #>= 1,  abs(X4-X7) #>= 2,  abs(X4-X8) #>= 1,
  abs(X5-X7) #>= 1,  abs(X5-X8) #>= 2,  abs(X6-X7) #>= 2,
  abs(X6-X8) #>= 1,  abs(X7-X8) #>= 2,
  labeling([ffc],L).
```

Figure 3: Prolog program to solve the L_{21}-colouring of 3×3 square grid.

3 Min-conflicts

The n-queens problem is well-known: we need to arrange n queens on a $n \times n$ board, that no one queen can attack any of the others. This problem was a benchmark for a long time, because it was hard to solve, since it contained many constraints: the figures cannot be in the same row, nor in the same column, and neither in the same diagonal. Suddenly, this problem became easy, a 10^6-queens problem could be solved within seconds with the method of minimal conflicts [4].

This method counts the violations of the constraints (conflict), and selects randomly chosen variables with such values for which the number of conflict is minimal. If the solutions are distributed uniformly within the search space, then this method quickly finds a solution, where the number of conflicts is zero.

We wrote in the introduction, that at map colouring problems this method works well. At the usual implementation the starting state is random, i.e. all nodes are assigned a colour randomly and independently from the colours of the other nodes. Next the program randomly select nodes and tries to reduce the number of conflict by recolouring the selected node.

The following question arises: does it works for distance-constrained colouring problems? The first column—with head *random/node*—of Table 1 shows,

that this method works poorly except for a few cases. In most of the cases it cannot solve the problem during hundreds of tests. The numbers in the rubrics denote the rate of the cases when the starting-state and the final-state—after the optimization—satisfied all the constraints.

Naturally the probability of randomly generating a solution is negligible. But the recolouring of the nodes according to the conflicts does not give solution, by our experiments. So it is worth to change the method of recolouring. Originally we recoloured only one node. At the variant we could recolour the whole neighbourhood. For this, at first we uncolour the selected node, and next we check all of its neighbours—if we have $L_{x_1 \ldots x_k}$ colourings, then all nodes whose distance is k or less from the selected node—, and if the colour of this neighbour can be decreased (without introducing conflicts), then we decrease. Finally we choose a colour for the selected node, for which the number of conflicts is minimal. If this whole step increases the number of conflicts, we restore the previous state.

[4] uses the 3-colouring of graphs as an example, and presents a method (Brelaz algorithm) which gives a good starting state. This method uses several considerations, which could be known from solving CSP.

The method begins with an uncoloured graph, and chooses a central node (with the most neighbour), and assigns the minimal colour to it. Next, the method repeatedly chooses an uncoloured node which has the minimum number of conflict-free (possible) colours. In case of a tie the maximal number of uncoloured neighbours determines the winner, or if this gives a tie again, then we can choose randomly from the best nodes. The selected node gets its minimal conflict-free colour.

The last columns in Table 1 denotes the minimal number of colours needed with this method without any constraint on number of colours and hence without conflicts. We typeset with bold the cases when it gives the chromatic number. As in this method a tie is very common, randomness has an important effect. The last columns show the rate of achieving this colouring.

4 Discussion

The second column (with head *random/neighbour*) of Table 1 contains bigger numbers than the preceding column. Hence the neighbour recolouring is advanced according to node recolouring. Unfortunately these numbers are small, even at bigger grids.

The Brelaz algorithm provides a compact colouring, i.e. the colours are very

size	dist.	random node	random neighbour	Brelaz neighbour	c.	r.
			square grid			
3×3	L_{21}	0.00→0.51	0.00→0.64	1.00→1.00	**7**	1.00
	L_{321}	0.00→0.19	0.00→0.64	0.70→0.80	11	0.31
4×4	L_{21}	0.00→0.04	0.00→0.07	0.81→0.81	**7**	0.67
	L_{321}	0.00→0.01	0.00→0.06	0.18→0.18	13	0.15
5×5	L_{21}	0.00→0.00	0.00→0.002	0.00→0.00	8	0.21
	L_{321}	0.00→0.00	0.00→0.002	0.00→0.00	14	0.32
			triangular grid			
3×3	L_{21}	0.00→0.00	0.00→0.011	0.00→0.00	9	0.47
	L_{321}	0.00→0.00	0.00→0.002	0.00→0.18	17	0.05
4×4	L_{21}	0.00→0.00	0.00→0.017	0.00→0.00	11	1.00
	L_{321}	0.00→0.00	0.00→0.00	0.00→0.00	21	1.00
			hexagon grid			
3×3	L_{21}	0.00→0.06	0.00→0.13	0.00→0.00	6	1.00
	L_{321}	0.00→0.25	0.00→0.41	0.33→0.33	**9**	0.32
4×4	L_{21}	0.00→0.03	0.00→0.11	0.05→0.14	**6**	0.10
	L_{321}	0.00→0.02	0.00→0.11	0.39→0.43	**10**	0.35
5×5	L_{21}	0.00→0.00	0.00→0.01	0.32→0.32	**6**	0.26
	L_{321}	0.00→0.00	0.00→0.02	0.00→0.00	**10**	0.23

Table 1: Success rate of the different min-conflicts variants. The column-heads denote the construction of the starting state (random/Brelaz) and the type of recolouring (node/neighbour). The last columns show the number of colours with Brelaz algorithm (c), and the rate of this colouring (r).

close to each other. There is some chance that the colouring will only use a chromatic number of colours, but it makes it almost impossible to recolour the nodes. Hence we can only see a few improvements in the third column of Table 1.

The distance-constrained colouring is a typical combinatorial optimization problem, at least in such sense that it has incredibly numerous local minima, hence the local optimization methods do not perform well, as the table shows. In case of optimization methods using crossover and mutation there is very little chance that by combining two independent individuals we get better individual with less conflicts. By our experiments the methods based on crowd (particle swarm optimization, harmony search, etc.) gave acceptable solutions

only for small grids, for bigger problem the crowd did not containe enough individuals to cope with the huge number of different colourings.

The big drawback of the optimization methods is that at the best case scenario it can only satisfy the existence of a k-colouring of the graph. Otherwise if the optimization method does not give a k-colouring in a fixed time, we *cannot* state, that for this k there does not exist a k-colouring.

The backtrack method gives more in this sense. It looks through the whole state space of the problem systematically. If this search ends without a solution, we can be sure, that the problem has no solution, i.e. the graph has no suitable k-colouring for a given k. The backtrack search in a strict sense is not an exhaustive search, because it does not check all the possible states. But it does not omit any state that can be a solution. Of course the heuristics are very important—because they can help to increase the number of states omitted, even in several orders of magnitude—as they help to discover, that a state is hopeless, i.e. cannot be a solution.

At finding the L_{21}-colouring of the 10×10 hexagonal grid the SWI-Prolog with the default setup found a solution in 311.2 seconds, and with the heuristics `ffc` even 0.07 seconds were enough. Of course as the grid is bigger, the rate of solving times becomes bigger, too.

The modern CPS systems contain many heuristics, so we can use them. If we only have a few colours, then the minimum remaining values' heuristic forces the backtrack many times, because the actual state violates some constraint. If we turn back from the blind alley early, then we perform faster, than if we go up to the walls.

As the number of colours usable at colouring increases this heuristic becomes weaker, it takes more steps and more hypotheses to determine unequivocally the colour of a node, or realize the blind alley at searching. In case of a 5×5 square grid we need to colour 25 nodes. To prove that 25 colours are not enough for the L_{4321}-colouring, the state space has 25^{25} nodes. By using heuristics we can omit most of these states, but a huge number of them remain to be checked, and this takes a lot of time.

We experimented with symmetry-breaking. As we are interested in the existence of the solution, and we need maximum one solution to present it, we can omit the rotated and mirrored solutions. So we can add hypothesis that the central node's colour is a low number (less or equal to the half of the maximal colour), among its neighbours the north one has the minimum value, etc. It is surprising, that finding a solution usually takes more time with this type of acceleration, than without it; the new constraints altered the direction of the search. But the exhaustive search became faster as we reduced the state

size	L_{21}		L_{321}		L_{4321}	
3×3	6:0.023	7:0.002	9: 0.040	10: 0.016	17: 1.574	18:0.544
4×4	6:0.033	7:0.012	11: 0.443	12: 0.017	22:179.563	23:6.235
5×5	6:0.051	7:0.023	11: 0.622	12: 0.036	25: ?	26:0.162
10×10	6:0.081	7:0.048	11: 1.407	12: 0.090	31: ?	32:2.418
20×20	6:0.183	7:0.187	11: 2.022	12: 0.354	35: ?	36:1.679
30×30	6:0.357	7:0.468	11: 2.971	12: 0.885	38: ?	39:5.392
100×100	6:3.398	7:8.009	11:21.673	12:11.819	-: -	-: -

Table 2: Solving time: constraint logic programming for square grids

size	L_{21}		L_{321}		L_{4321}	
3×3	7:0.044	8:0.048	14: 1.147	15:0.461	21:9.544	22:3.179
4×4	8:0.213	9:0.016	16:12.770	17:0.091	26: ?	27:0.167
5×5	8:0.280	9:0.027	17:39.704	18:0.062	29: ?	30:2.511
10×10	8:0.440	9:0.071	23: ?	24:0.169	41: ?	41:0.421
20×20	8:0.727	9:0.282	23: ?	24:1.131	44: ?	45:2.763
30×30	8:0.143	9:0.619	25: ?	26:4.685	46: ?	47:9.072

Table 3: Solving time: constraint logic programming for triangle grids

space to its one eighth in the case of square grids. For example 946 seconds were enough to show that there is no L_{4321}-colouring of 5×5 square grid with 25 colours. This can extend our barriers, but the cases with many colours are hopeless with this method.

The Tables 2–4 shows the time needed to *prove* that for number k there is no k-colouring; and what the search time for a n-colouring is. One cell contains k with proof-time, and n with search-time. If $k+1 = n$, then number n is the chromatic number, because we *proved* that less colours are not enough.

At some cases—denoted with question mark—the time needed for the proof is not known. If k is much smaller than the chromatic number of the graph, then it is easier to violate the constraints, or with other words it is harder to satisfy them. So we got into contradiction much earlier, and even the search space is smaller, and we prove the uncolourability much faster. If n is slightly bigger than the chromatic number, the search time increases as n increases. As we have more and more opportunities to assign a colour to a node, it takes more time to check more cases. If n is much larger than the chromatic number,

size	L_{21}		L_{321}		L_{4321}	
3×3	4:0.009	5:0.009	8:0.022	9:0.009	13: 0.135	14:0.111
4×4	5:0.013	6:0.011	9:0.126	10:0.012	17: 5.710	18:6.724
5×5	5:0.015	6:0.010	9:0.114	10:0.014	19:71.029	20:1.251
10×10	5:0.034	6:0.046	9:1.185	10:0.226	24: ?	25:4.094
20×20	5:0.152	6:0.080	9:2.349	10:1.335	29: ?	30:1.212
30×30	5:0.162	6:0.335	9:2.603	10:4.440	31: ?	32:4.431

Table 4: Solving time: constraint logic programming for hexagon grids

we need less backtracking because more numbers allow satisfying constraints easily, but in the case of backtracking we need to check more cases. By the experiments these two effects compensate each other, and the search times are similar.

For big grids the source with constrains can even take megabytes. Interesting, that for grids with small chromatic number the solving of the problem takes less time than loading into memory and compiling. In the case of L_{4321}-colouring of the 100×100 we did not have enough memory to run the search.

5 Conclusion and future work

In this article we have examined two methods for solving distance-constrained colouring. As colouring in general is a hard task, it was expected that there would be no royal road. The performance of constraint (logic) programming is good for graphs with a small chromatic number. If the chromatic number is high, we could get a solution within a short period of time for a nearby number, but we cannot be sure whether this number is equal with the chromatic number, or not. If there are too many constraints, then the solver cannot solve the problem, due to the shortage of the memory.

Although the method of minimal conflicts has many nice results for solving optimization problems, we were not able to apply it for the distance-constrained colouring. The method needs more fine-tuning, to make bigger realignment of colours in one optimization step.

We are planning to test other optimization methods on this type of problems, in hope of finding a more effective method.

The results of our experiments raise some questions. As Table 2 and 4 shows in case of square and hexagon grids the chromatic number for colouring L_{21} and

L_{321} is constant over a minimal size. Table 3 shows constant chromatic numbers for L_{21}-colourings. Does there exist a constant value in this case for L_{321} or not? Moreover are there constant values for L_{4321}-colourings for these kind of grids? The tables in this column contain many question marks, denoting that the exhaustive search needs a long time (and we had no patience, to wait for the exact time). Our experiments are not enough to answer this question now.

It is obvious, that if we have a colouring of an $n \times m$ grid then we can truncate from it a colouring of a $k \times l$ grid, where $k \leq n$ and $l \leq m$. Can we repeat the first row and column of a 608 different L_{21}-colouring of the 3×3 square grid to get the L_{21}-colouring of the 4×4 square grid? Or in more general: can we rotate, mirror or translate one colouring of a small grid, or combine several colourings of the same small grid to produce colouring of a bigger grid?

Does there exist a circular colouring of the square grids (by treating them as a torus), which can be extended into a colouring of an infinite grid? In this case are the chromatic numbers different, or not?

Acknowledgements

Many thanks for Gábor Halász for valuable questions and remarks.

References

[1] T. Calamoneri, The $L(h,k)$-labelling problem: A survey and annotated bibliography, *The Computer Journal* **49**, 5 (2006) 585–608. ⇒7

[2] P. Galinier, J. K. Hao, Hybrid evolutionary algorithms for graph coloring, *J. Comb. Optim.*, **3**, 4 (1998) 379–397. ⇒6

[3] W. K. Hale, Frequency assignment: Theory and applications, *Proc. of the IEEE* **68**, 12 (1980) 1497–1514. ⇒6

[4] S. Minton, et al. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, *Artificial Intelligence* **58,** 1, (1992) 161–205. ⇒6, 9, 10

[5] P. Panigrahi, A survey on radio k-colorings of graphs, *AKCE Int. J. Graphs Comb.* **6**, 1 (2009) 161. ⇒7

[6] S. J. Russel, P. Norvig, *Artificial intelligence: a modern approach*, Pearson 2002. ⇒6, 8

Received: January 31, 2016 • Revised: March 8, 2016

DE GRUYTER OPEN
Acta Univ. Sapientiae, Informatica **8,** 1 (2016) 16–40

DOI: 10.1515/ausi-2016-0002

A survey of the all-pairs shortest paths problem and its variants in graphs

K. R. UDAYA KUMAR REDDY
Department of Computer Science and Engineering
NMAM Institute of Technology
Nitte–574 110, India.
email: krudaykumar@nitte.edu.in

Abstract. There has been a great deal of interest in the computation of distances and shortest paths problem in graphs which is one of the central, and most studied, problems in (algorithmic) graph theory. In this paper, we survey the exact results of the static version of the all-pairs shortest paths problem and its variants namely, the Wiener index, the average distance, and the minimum average distance spanning tree (MAD tree in short) in graphs (focusing mainly on algorithmic results for such problems). Along the way we also mention some important open issues and further research directions in these areas.

1 Introduction

1.1 Motivation

The problem of finding the shortest distance between two vertices of a graph and finding a path that causes it are classic problems in graph algorithms. It appears in countless practical applications and is an important concept in transportation (and communication) engineering, computer science, network routing, network analysis [63], image processing [37, 61], operation research [19, pages 657], VLSI design [66], DNA analysis [70], bio-informatics

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 68R10
Key words and phrases: algorithms, all-pairs shortest paths, average distance, graph algorithms, MAD tree, spanning tree, Wiener index.

[39], chemical compounds [6, 31], computational geometry and robotics [33], to mention few central areas of interest. Because of its rich in applications, the work on such problems are deep and vast (in all kinds of classic graphs, directed or undirected, weighted or unweighted), in both the scientific community and engineering community. In addition, shortest path algorithms also have applications as a subroutine in other combinatorial optimization algorithms such as network flows [19, pages 708–709]. One of the basic and key problem in transportation and network analysis is the computation of the shortest paths between any two locations on a network. In computer network, the transportation could be routing messages. For instance, given a road map (or network) on which the distance is marked between every pair of adjacent cities (or nodes), find the shortest possible route between every such pair of cities (or nodes). For such a road map, we can model the graph by representing cities as vertices, road segments between cities as edges, and road distances as edge weights.

1.2 Preliminaries and notations

Let $G = (V(G), E(G))$ be a connected and simple (i.e., without loops and multiple edges) graph on $|V(G)| = n$ and $|E(G)| = m$. The graph G may be directed or undirected, and edges of G may be weighted or unweighted. If G is weighted, then the edge weights may be real-valued or integers, of either negative or nonnegative. For a weighted graph, the weight of a path is the sum of the weights of its edges on that path. For an unweighted graph, the weight of an edge is taken to be one. A shortest path between two vertices u and v is a path of minimum weight. For $u, v \in V(G)$, the distance between two vertices u and v, denoted by $d(u, v)$ is defined as follows. (i) Graph G is unweighted: The distance $d(u, v) = 1$, if $uv \in E(G)$, and $d(u, v) =$ the length of a shortest path joining u and v (or smallest number of edges connecting u and v), otherwise. (ii) Graph G is weighted: Here the distance $d(u, v) =$ the sum of the weights of the edges along the shortest path joining u and v.

The single-source shortest distance (SSSD) problem is to compute $d(s, v)$ from a given source vertex s to all other vertices v in the graph. The single-source shortest path (SSSP) problem is to compute the shortest paths from a given source vertex s to all other vertices in the graph. The all-pairs shortest distance (APSD) problem is to compute $d(u, v)$ between all pairs of vertices u, v in the graph. The all-pairs shortest path (APSP) problem is to compute the shortest paths between all pairs of vertices in the graph.

Given an unweighted undirected graph G, the Wiener index $W(G)$ of G is

defined as

$$W(G) = \frac{1}{2} \sum_{u \in V(G)} \sum_{v \in V(G)} d(u, v). \tag{1}$$

The Wiener index comes under various names such as transmission, total status, gross status, graph distance, and sum of all distances.

For an unweighted undirected graph G, the average distance $\mu(G)$ of G is defined by

$$\mu(G) = \frac{1}{n(n-1)} \sum_{u \in V(G)} \sum_{v \in V(G)} d(u, v) = \frac{2W(G)}{n(n-1)}. \tag{2}$$

From (2), we see that the quantity $\mu(G)$ is closely related to $W(G)$. For a connected undirected graph G, with a nonnegative edge weight, the Minimum Average Distance (MAD) spanning tree of G (MAD tree in short) is a spanning tree of G with minimum average distance.

The Wiener index of vertex-weighted graphs was introduced in [54]. If G is a graph with weight function $w : V(G) \to \mathbb{N}^+$, then the Wiener index $W(G, w)$ of a vertex-weighted graph (G, w) is defined as

$$W(G, w) = \frac{1}{2} \sum_{u \in V(G)} \sum_{v \in V(G)} w(u) w(v) d(u, v). \tag{3}$$

Notice that if all weights of vertices in the graph are unit (or one), then $W(G, w) = W(G)$.

It is well known that there are many variants of shortest paths problem. Typically it is categorized into the SSSP problem and the APSP problem. In this short survey we consider only the APSP problem and their related problems in graphs.

A survey by Zwick [90], the exact and approximate distances in graphs are considered. In particular, the paper gives a survey on both SSSP and APSP algorithms and the related distances such as spanners (a sparse subgraph that approximates all the distances between every pair of vertices) and approximate distance oracles (concise representation of approximate distances together with quick means of extracting these approximations). Although [90] gives a survey on the APSP problem and mentions some open problems, we revisit this problem in detail and in addition we survey the recent results on general graphs as well as on restricted family of graphs. Furthermore we also give the survey on the variants of APSP problem defined above mostly focussing on the results of algorithmic computation.

We begin with the all-pairs shortest paths problem.

2 All-pairs shortest paths problem

2.1 Background

The APSP problem is undoubtedly one of the most fundamental and classical problem in graph algorithms and is well known in the research community, yet, the complexity of this problem has remained open to date even though it runs in polynomial time. Recall that the significance of the APSP problem and each of its variants defined above are rich in applications. Almost all known shortest path algorithms are computed based on the following two types of computational models:

- Comparison-addition model: The shortest path algorithms in this model assumes the input to be real-weighted graphs, where the only operations allowed on reals are comparisons and additions and no other operations are allowed. These operations are assumed to take $O(1)$ time. The comparison operation finds the larger of two real numbers, whereas the addition operation generates a new real number from the existing two real numbers. Clearly, in this model, based on the outcome of previous comparisons, the algorithm chooses the next operation. Moreover, since this model fits with the assumption of real numbers which are arbitrary values, it cannot distinguish between reals and integers–hence no integer can be produced from a real variable.

- Random Access Machine (RAM) model: Here the shortest path algorithms assume the input to be integer-weighted graphs, where integers are manipulated by additions, subtractions, comparisons, shifts, and various logical bit operations on machine words (see [1]). These operations take $O(1)$ time. In this model, each word of memory is assumed to be w-bit wide, capable of holding an integer in the range $\{-2^{w-1}, \ldots, 2^{w-1}-1\}$. Usually, it is assumed that $w = \Theta(\log n)$, which is the standard realistic assumption, where n is the input size.

Note that most of the intricate algorithms solving shortest paths problem, work in RAM model. Given the APSP problem, it is important for us to consider the implementation of this algorithm in practice. But unfortunately, some of the algorithms are far from being practical. So, there is a great deal of interest in obtaining faster algorithms in solving the APSP problem on general graphs in practice. The progress on the APSP problem have focused mostly on the following two approaches:

- Combinatorial approach: Combinatorial algorithms are good in practice because they rely on the efficient computation of small subproblems, but unfortunately these algorithms are typically worse in theoretical bounds. However these algorithms can give better asymptotic bounds for sparse graphs.
- Algebraic approach: Algebraic algorithms are far from being practical because they suffer from large hidden constants in the running time bound and large overhead of fast matrix multiplication since they rely on matrix multiplication over a ring, but fortunately these algorithms achieve good theoretical bounds. Hence these algorithms may be viewed of only theoretical interests.

Finding the APSP problem in a graph has received a considerable attention from the research community and have been studied extensively in both theory and practice. Although the APSP problem is solvable by repeated applications of SSSP algorithms, the APSP problem can be solved efficiently for arbitrary graphs as well as for other classes of problems. The various class of problems include: algorithms for restricted families of graphs, such as interval graphs (see [3, 57, 68, 75]), circular arc graphs (see [3, 75]), strongly chordal graphs (see [5, 20, 44]) etc.; algorithms for dynamic versions of the problems (see [28, 53]); parallel algorithms (see [16, 17]). For certain applications, computing exact distances (respectively shortest paths) over all pairs of vertices of a graph may be quite expensive. Also the algorithms computing exact distances using fast matrix multiplication technique are impractical since it suffers from large hidden constants. In the last decade many researchers considered all-pairs approximate shortest paths (APASP) problem [74] where a number of subcubic algorithms have been designed using a simple and novel combinatorial ideas, and also designing an optimal (quadratic) algorithms for some restricted graph classes using simple and efficient approach. Optimal algorithms for some restricted graph classes are discussed in Section 2.4. Often in practice, instead of computing exact distance (respectively shortest paths) between any pair of vertices of a graph, computing near distance (respectively shortest paths) is good enough. An algorithm computing approximate distance (or shortest path) for any given pair of vertices may have some kind of error associated with the distance (or shortest path) and this error can be additive (also known as surplus) or multiplicative (also known as stretch). That is, approximate distances (or shortest paths) are longer than the actual distances (or shortest paths). The all-pairs surplus-a length, $a \geq 0$, is to compute the length of at least $d(u, v)$ and at most $d(u, v) + a$ for all pairs of vertices $u, v \in V(G)$.

The all-pairs stretch-t length, $t \geq 1$, is to compute the length of at least $d(u, v)$ and at most $t \cdot d(u, v)$ for all pairs of vertices $u, v \in V(G)$.

Statement of the APSP problem: Given an input graph $G = (V(G), E(G))$, the goal is to compute the distances, and construct their corresponding shortest paths between all pairs of vertices in the graph. The output is thus a distance matrix and the shortest paths that causes it.

In the following subsections, only the exact results of the static version of the APSP problem are considered.

2.2 Arbitrary weighted graph

Suppose the problem needs to report the distances (respectively shortest paths) between all pairs of vertices of the graph. Then the time complexity for such a problem must be at least $\Omega(n^2)$ since there are $\binom{n}{2} = \Theta(n^2)$ pairs of vertices. It is well known that the straightforward approach to solve the APSP problem is to run the SSSP problem for each vertex (as source) of the graph. The classical results of SSSP problems are Bellman-Ford and Dijkstra's algorithms. Using Bellman-Ford SSSP algorithm n times [19, 651–654], the APSP problem for arbitrary graphs with real-edge weights can be solved in time $O(n^2 m)$. Using Dijkstra's SSSP algorithm n times [27], and using a Fibonacci-heap implementation Fredman and Tarjan [40] (with Johnson's [51] preprocessing step if negative weights are allowed), the APSP problem (on nonnegative edge weights) can be solved in time $O(n^2 \log n + mn)$. Then Pettie [64] has shown that the first term can be reduced to $O(n^2 \log\log n)$ time. As m can be as high as $\Omega(n^2)$, the running time of the above algorithms can be as bad as $\Omega(n^3)$. However, the APSP problem can be solved efficiently without the application of SSSP algorithms. It may be noted that all of the algorithms aforementioned in this subsection work in comparison-addition model.

2.2.1 Dense real-weighted graphs

The classical Floyd-Warshall algorithm [19, 693–697] solves the APSP problem using dynamic programming technique in time $O(n^3)$. It is one the notable algorithms that is too short, simple to implement, well understood, and works better on adjacency matrices than adjacency lists. Also note that this algorithm work in comparison-addition model. Let $A = (a_{ij})$ and $B = (b_{ij})$ be two $n \times n$ matrices. Then the distance matrix multiplication (DMM) $C = AB$

is an $n \times n$ matrix $C = (c_{ij})$ with $c_{ij} = \min_{k=1}^{n}\{a_{ik} + b_{kj}\}$ for $1 \leq i, j \leq n$. The distance matrix multiplication (also known as the min-plus matrix multiplication) can be naively computed using $O(n^3)$ additions and comparisons. It is well known that the APSP problem is closely related to the problem of computing distance product of matrices. That is, the time complexity of DMM is asymptotically equal to that of the APSP problem for a graph with n vertices (see [1]). Fredman [41] was the first to find the possibility of obtaining a subcubic algorithm for computing the distance product of two $n \times n$ matrices using only $O(n^{2.5})$ comparisons and additions. But there is no clear specification about as to which comparisons should be made nor how to infer the result from the outcome of these comparisons; however, Fredman was able, to use his observation to obtain an explicit subcubic algorithm for the distance product, and hence for the APSP problem. His approach was based on the construction of decision tree whose depth is $O(n^{2.5})$ to solve certain small-sized subproblems. However there is no known polynomial-time implementation of his algorithm. Fredman [41] somehow was able to show that the APSP problem can be solved in $O(n^3 \log^{1/3} n / \log^{1/3} n)$ time. Takaoka [77] presented much simpler and efficient algorithm in $O(n^3 \log^{1/2} n / \log^{1/2} n)$ time based on similar ideas but using table look-up. Dobosiewiczs [30] obtained $O(n^3 / \log^{1/2} n)$ time algorithm by exploiting a different approach bit-level parallelism (i.e., simultaneous operations on $\log n$ bits contained in a single machine word). Han [46], Takaoka [78], and Zwick [92] obtained, respectively, $O(n^3 \log^{5/7} \log n / \log^{5/7} n)$, $O(n^3 \log^2 \log n / \log n)$ and $O(n^3 \log^{1/2} \log n / \log n)$ time algorithms and they all involved even more complicated combinations of approaches. Chan [10] obtained $O(n^3 / \log n)$ time algorithm by using somewhat different approach. The speedup of his algorithm is obtained by using a simple geometric approach. Han [47] obtained $O(n^3 \log^{5/4} \log n / \log^{5/4} n)$ by using a more sophisticated word-packing tricks. Chan [12] obtained $O(n^3 \log^3 \log n / \log^2 n)$ time algorithm using the approaches from the previous algorithms by Chan [10] and by Han [47]. That is, his algorithm combines the geometric approach by Chan [10] and more sophisticated word-packing tricks by Han [47]. The paper also gives the results for APSP problem on a large class of geometrically weighted graphs, where the weight of an edge is a function of the coordinates of its vertices. His approach also extends to the case of small-integer-weighted graphs which is not as good as Zwicks algorithm [91]. However, his approach more generally extends to the case where weights are taken from small set of real values yielding the first truly subcubic result ($O(n^{2.844})$ time) for APSP in real vertex-weighted graphs, as well as an improved result ($O(n^{2.688})$ time) for the all-pairs lightest shortest path (among all shortest paths connect-

ing u and v, the lightest shortest path is a shortest path that uses smallest number of edges) problem for small integer-weighted graphs. As Tong-Wook Shinn and Tadao Takaoka mentions in [72], currently the best known upper bound is $O(n^3 \log\log n / \log^2 n)$ due to Han and Takaoka [48].

2.2.2 Dense integer-weighted graphs

Matrix multiplication is one of the most basic problems in mathematics and computer science. For a long time, the fastest algorithm for multiplying two $n \times n$ matrices was $O(n^\omega)$ time bound, where $\omega < 2.376$ [18]. Recently, Vassilevska Williams [84], improved this long-standing bound to $O(n^\omega)$ time bound, where $\omega < 2.3727$. Very recently, Le Gall [56] achieved a faster algorithm which is better than all known algorithms for rectangular matrix multiplication. Thanks to Le Gall [56] for his achievement in rectangular matrix multiplication. The results obtained in [56] are a generalization of Coppersmith-Winograds approach [18] to the rectangular setting. Moreover the algorithms presented in [56] gives an improvement for computing the product of two sparse square matrices but for the product of dense square matrices. Thus we assume that the fastest algorithm for multiplying two $n \times n$ matrices in general is $O(n^{2.3727})$ time due to [84]. In the last decade, many researchers have shown that matrix multiplication over rings [18, 76] can be used to obtain faster algorithms (i.e., subcubic algorithms) for solving the APSP problem in dense graphs with small integer edge weights, where the weights lie in the range $\{1, \ldots, M\}$ for some constant M. In this case, Galil and Margalit [42] gave an $\tilde{O}(M^{(\omega+1)/2} n^\omega)$ time algorithm for undirected graphs, and Alon et al. [2] gave an $\tilde{O}(n^{(3+\omega)/2}) = O(n^{2.688})$ time algorithm for directed graphs. Then a series of subcubic algorithms [42, 71, 73, 91] have been developed. Currently, the best known algorithm for the APSP problem over directed graphs with small integer weights is in time $O(n^{2.5302})$ due to [56] (since it relies on rectangular matrix multiplication), improving over $O(n^{2.575})$ time by Zwick [91]. For undirected graphs, currently the best known APSP algorithm using fast matrix multiplication technique is $\tilde{O}(Mn^\omega)$ time bound due to Shoshan and Zwick [73].

2.2.3 Sparse graphs

Johnson [51] observed that the APSP problem can be solved efficiently on sparse graphs. That is, if the graph has negative edge weights with no negative-weight cycles, then the new set of nonnegative edge weights allows to pre-

serve shortest paths by reweighting technique (see [19, pages 700–704]) in time $O(mn)$. Johnson's algorithm uses the Bellman-Ford and Dijkstra's algorithms as subroutines yielding an APSP algorithm for arbitrary weighted graphs in time $O(mn + n^2 \log n)$. Later, Pettie [64] has shown that the second term can be reduced to $O(n^2 \log\log n)$ time for directed graphs, and Pettie and Ramachandran [65] has shown that the second term can be reduced to $O(n^2\alpha(m, n))$ time for undirected graphs, where α is slow growing inverse-Ackermann function.

There are also results for solving the APSP problem on integer-weighted graphs. The existing implementations of Dijkstra's algorithm on APSP [45, 81] for integer-weighted graphs run in time $O(\min\{mn\ (\log\log n)^{1/2},\ mn + n^2 \log\log n\})$. Then Thorup [80] gave bounds on APSP problem using hierarchy-based approach for undirected graphs in time $O(mn)$. Later Hagerup [43] was able to show the bounds on APSP problem for directed graphs using hierarchy-based approach in time $O(mn + n^2 \log\log n)$. The author generalized Thorup's approach that gave a new approach to view the commonalities between all hierarchy-type algorithms and in particular, it gives a one-line characterization for all hierarchy-type algorithms. This characterization leads to prove lower bounds on their complexities in the comparison-addition model.

2.3 Arbitrary unweighted graph

2.3.1 Dense graphs

For a weighted graph G, let the edge weights of G be a unit weight. Then, the APSP problem can be solved by the simple Floyd-Warshall algorithm [19, pages 693–697] in time $O(n^3)$. Improved results show that it runs in $O(mn + n^2 \log n)$ time (Dijkstra's algorithm [19, pages 658–662], Johnson [51], Fredman and Tarjan [40]). Using Pettie algorithm [64] the APSP problem can be solved in time $O(n^2 \log\log n + mn)$. Taking G to be an unweighted graph, the algorithm for APSP can be solved by running breadth-first search (BFS) [19, pages 594–601] once from each vertex (as source) of G. This takes a time $O(n^2 + mn)$. As m can be as high as $\Omega(n^2)$, the running time of the above algorithms can be as bad as $\Omega(n^3)$. Again all of the algorithms aforementioned in this subsection work in comparison-addition model.

For unweighted undirected graphs, Galil and Margalit [42] and Seidel [71] showed that fast matrix multiplication algorithms can be used to obtain improved algorithms for the APSP problem for graphs with small integer weights. The running time of their algorithms is $\tilde{O}(n^\omega)$ (it may be noted that $\tilde{O}(n)$

$= O(n \log^k n$; that is, $\tilde{O}(n) = O(n)$ with logarithmic factors ignored). The Seidel's algorithm is simple and elegant. However, there seems to be no simple way of using Seidel's elegant technique on weighted graphs. The algorithm of Galil and Margalit [42], on the other hand, can be extended to handle on weighted (small integer weights) graphs.

There are also results for unweighted directed graphs for solving the APSP problem that use fast matrix multiplication algorithms when the edge weights are small integers. In this case, Alon et al., [2] were the first to obtain a $O(n^{2.688})$ algorithm. The authors first give a simple version of the APSP problem for directed graphs with edge weights taken from the set $\{-1, 0, +1\}$. Then they have extended their algorithm to the case of edge weights which are integers with a small absolute value. Later Zwick [91] obtained a running time of $O(n^{2.575})$. Recently, Le Gall [56] has improved the result to obtain a running time of $O(n^{2.5302})$.

By exploiting graph compression technique, a new combinatorial idea, Feder and Motwani [38] obtained $O(mn \log(\frac{n^2}{m})/\log n)$ time, an improved algorithm that achieves log-factor speedups for solving APSP in the unweighted undirected graphs. Generally, log-factor speedups may be possible when there is some amount of redundancy or repetition in the input or computational process. As mentioned in [38], the compressed graph G^* of a graph G encodes some aspects of the structure of G and has the following properties: (i) G^* is a graph with m^* edges, where $m^* < m$. (ii) It is computationally easy to convert G into G^*, and vice versa. Therefore, the compression may be viewed as a data structure for representing the graph G.

2.3.2 Sparse graphs

Recall that the straightforward approach for solving APSP problem is to run BFS once from each vertex (source) of G which takes a time $O(n^2 + mn)$. The algorithms of Galil and Margalit [42], and Seidel [71] do not improve over the naive approach $O(n^2 + mn)$ algorithm when $m = o(n^{\omega-1})$. As mentioned above, Feder and Motwani [38] obtained $O(mn \log(\frac{n^2}{m})/\log n)$ time algorithm which is an improvement over $O(mn)$. Then, Chan [11] gave bounds that runs in $o(mn)$ time for undirected graphs. This analysis is based on one crucial parameter, m, the number of edges in the graph. That is, the results for small values of m. In [11] the time bounds of the algorithms improves over time bounds including the naive bound, Seidel's [71] algorithm, and Feder and Motwani's [38] algorithm, for all $m \ll n^{1.376}$. Later, Blelloch et al. [7] presented a new combinatorial data structure method that beats Feder and Motwani's and

Chan's result. The running time of their algorithm is $O(mn \log(\frac{n^2}{m})/\log^2 n)$. The ability of this new data structure lies in computing sparse vector products quickly and tolerate matrix updates. Using their data structure, the authors give best running time bounds for four fundamental graph problems: transitive closure, APSP on unweighted graphs, maximum weight triangle, and all pairs least common ancestors. The authors also point out that by using the data structure gives the first asymptotic improvement over $O(mn)$ for all pairs least common ancestors on directed acyclic graphs.

2.4 Restricted family of graphs

When dealing with special graph classes, it is well known that the algorithmic computation of the given problem on such graph classes can be solved more efficiently. These algorithms are good enough for many practical applications since graph classes are more structured than just being general graphs. Computing exact distances (respectively shortest paths) over all pairs of vertices of a graph may be quite expensive on general graphs. So, many researchers started considering the APSP problem on special graph classes. The results on some of the restricted classes of graphs are truly overwhelming because of some optimal $O(n^2)$ algorithms are known for solving the APSP problem. These include: interval graphs, circular arc graphs, planar graphs (see [49]), permutation graphs, bipartite permutation graphs, strongly chordal graphs, chordal bipartite graphs, distance-hereditary graphs, and dually chordal graphs. The results of APSP problem for such graph classes [33] may be consulted.

The work by Han et al. [44], presents an optimal $O(n^2)$ time algorithm for solving the APSP problem on chordal graph if G^2 ($G^2 = (V, E')$, where $\{u, v\} \in E'$ if and only if $1 \leq d(u, v) \leq 2$) is known. The authors claims that computing G^2 for chordal graphs is as hard as for general graphs. They also point out that G^2 can be computed more efficiently for special classes of chordal graphs such as planar chordal, k-chordal, and strongly chordal giving rise to optimal algorithms for the APSP problem on these classes of graphs in a more natural way than the previous results. An optimal parallel algorithm for the APSP problem on chordal graphs are also presented.

The author in [33], gives a simple and efficient approach to solve all-pairs approximate shortest paths (APASP) problem on the class of weakly chordal graphs and its subclasses. Moreover, the work in [33] presents a unified approach to solve APASP and APSP problems on graph classes including chordal, strongly chordal, chordal bipartite, and distance-hereditary graphs. A few open problems related to the distances are also suggested. Later, Mondal et al. [59]

and Saha et al. [69] obtained optimal algorithms for solving APSP on the class of trapezoid graphs and circular arc graphs respectively. For the definition of various families of graphs [8, 33] may be consulted.

An important observation: For each of the graph classes aforementioned, the APSP problem is solved using a simple and efficient approach which are important from the practical point of view.

2.5 Concluding remarks and open issues of APSP problem

Based on the results aforementioned, we conclude this section with few open issues and remarks on computing the APSP problem.

- Whether in general there exists a truly sub-cubic algorithm for the APSP problem in the comparison-addition model that runs in time $O(n^{3-\varepsilon})$, for some constant $\varepsilon > 0$?

- On an arbitrary unweighted graphs, the fact that the fastest combinatorial algorithm for APSP problem (despite aforementioned fast non-combinatorial algorithms based on matrix multiplication) is by running BFS [19, pages 594–601] once from each vertex (as source) of a graph. Obtaining a faster combinatorial algorithm for APSP problem on such graphs in fact will be a major breakthrough.

- Recall that the APSP problem is closely related to the problem of computing distance product of matrices. Thus the most important open issue in algebraic complexity, is finding the optimal value of the exponent of square matrix multiplication. For further information on square matrix multiplication we refer [56] and references therein.

- The author in [43] gave the bounds on APSP problem for directed graphs in the word RAM model that runs in time $O(mn+n^2 \log\log n)$. It would be desirable to obtain the bound $O(mn)$.

- For undirected graphs, currently the best known APSP algorithm using fast matrix multiplication technique is $\tilde{O}(Mn^{\omega})$ time bound due to Shoshan and Zwick [73]. As mentioned in [90], the authors in [73] show that the APSP problem for undirected graphs with edge weights taken from $\{0, 1, \ldots, M\}$ is harder than the problem of computing the distance product of two $n \times n$ matrices with elements taken from the same range by at most a logarithmic factor. Thus on undirected graphs a challenging problem for the APSP problem could be, by considering larger values of M, and obtaining truly sub-cubic algorithm for it.

- A great variety of optimal ($O(n^2)$) algorithms were developed for solving the APSP problem on restricted family of graphs as mentioned in section 2.4. It would be interesting to know for which other greater graph classes the APSP problem can be solved in $O(n^2)$ time.

Next we consider the Wiener index or average distance problem.

3 Wiener index or average distance

3.1 Background

The *Wiener index* (or *Wiener number*) problem is a well-known distance based graph invariant in mathematical chemistry. The work on the theory of Wiener indices is deep and vast, in both the biochemical community and mathematical community. In chemical graph theory, the structure of a chemical compound is usually modeled as a polygonal shape–paths, trees, graphs, etc., which is often called the *molecular graph* of this compound, where each vertex represents an atom of the molecule, and covalent bonds between atoms are represented by edges between the corresponding vertices (see [4, 67, 82]). In chemistry, much of the problems have influenced the development of graph-theory-based molecular structure-descriptors called the *topological indices*. Among all the topological indices, Wiener index $W(G)$ is the most important one (from middle 1970s), thoroughly studied and frequently used (see [87]). The Wiener index have been studied in both the mathematical and chemical literature. The majority of chemical applications lie in the study of Wiener index of acyclic (molecular) graphs. Most of the prior work on Wiener indices deals with two types of problems: Wiener index problem for graphs and the inverse Wiener index problem. The Wiener index problem deals with the efficient computation of the index, the upper and lower bounds on the index values, and the relation of the Wiener index to other quantities of the graph. The inverse Wiener index problem is: Given a positive integer k, does there exist a graph whose Wiener index is k. If so, can we compute it efficiently? For more information on the Wiener index especially for trees see [31] and references therein.

The *average distance* is one of the important parameter in metric graph theory. As mentioned in [55], it has numerous applications in many areas including computer science, cheminformatics, mathematics, and recent application in phylogenetics (see [88]). One of the fundamental parameter in computer science is to measure the cost of communication in a computer net-

work. In a network model, the time delay or signal disgradation in sending a message between any two points is often proportional to the distance a message must travel. When G represents a network, the average distance $\mu(G)$ problem can be viewed as a tool in analyzing networks since it is a measure on the time needed to traverse the messages between two randomly chosen points in the average-case performance of a network as opposed to the diameter (maximum of all shortest-path distances), which indicates the worst-case performance time [23]. Therefore, the quantity $\mu(G)$ play a significant role in analyzing communication networks and has been studied widely in [9, 23, 24, 25, 26, 29, 62, 88]. For more information on the average distance see [9, 62] and references therein.

3.2 Computation of Wiener index or average distance

The previous and ongoing work related to Wiener index or average distance are: Wiener index problem for graphs and the Inverse Wiener index problem. As a result, the research findings on such problems include obtaining upper and lower bounds, determining relationships to other quantities, determining inverse Wiener index problem, obtaining closed form expressions, and algorithmic determination of the Wiener index. In this short survey we discuss only a few important closed form expressions and the algorithmic determination of the Wiener index or the average distance.

One motivation comes from the following problem posed by F. R. K. Chung [15]: Is there an asymptotically faster algorithm for computing average distance than computing all distances between vertices of the graph?

S. Klavžar *at el.* [55], mentions that the average distance can be studied equivalently as the Wiener index (or the network distance)–hence the fundamental task would be the computation of the average distance or the Wiener index efficiently. For a general graph G, one way of computing $W(G)$ or $\mu(G)$ algorithmically is to run the APSD problem; that is it can be computed in polynomial time, [58]. Thus the results of APSP problem for unweighted undirected graphs including arbitrary graphs and restricted classes of graphs follows for computing $W(G)$ or $\mu(G)$. Besides, more efficient algorithms (i.e., linear or even sub-linear algorithms) have been developed for its computation on some restricted classes of graphs. In [13] a linear algorithm were proposed for the Wiener index of benzenoid graphs, while in [14] a sub-linear time algorithm for the same problem were proposed. Some of the results on restricted classes of graphs are mentioned below.

As Dobrynin *et al.* mentions in [31], the early work by Entringer *et al.*

[34], closed form expressions for $W(G)$ for large classes of trees are given. Among all trees of order n, the best known are $W(P_n) = \binom{n+1}{3}$ and $W(S_n) = (n-1)^2$, where P_n and S_n denote the path and star of a graph of order n respectively. It is easy to see that among all trees of order n, the Wiener indices of P_n and S_n, respectively, are maximum and minimum. In [34] it has been showed that for any tree T of order n that is different from P_n and S_n, $W(S_n) < W(T_n) < W(P_n)$. Dobrynin et al. [31], gives a detailed survey on the results known for the Wiener index of different class of trees. For example, the authors outlines computational methods of $W(G)$, combinatorial expressions for $W(G)$, connections between $W(G)$ and the center and centroid of a tree, and connections between $W(G)$ and the Laplacian eigenvalues. Results on the Wiener indices of line graphs of trees, on trees extremal w.r.t. $W(G)$, and on integers which cannot be Wiener indices of trees are also given. The related theory and applications are also mentioned.

Hexagonal systems (HS's) are a special type of plane graphs in which all interior regions (faces) are bounded by hexagons; that is, the two hexagons either have one common edge or have no common vertex, and no three hexagons share a common edge. HS's have applications in chemistry since they provide a graph representation of benzenoid hydrocarbons. In [32], the authors outlines the results known for Wiener index W of the HS: method for computation of W (W of a HS can be computed in time $O(n)$ and a sublinear time algorithm for simple HS's), expressions relating W with the structure of the respective HS, results on HS's extremal with respect to W, and on integers that cannot be the W-values of HS's.

An interesting conjecture on Wiener index of trees: It states that except for some finite set of integers, every integer n is the Wiener index of some tree. Ban et al. [6], showed that enumerating all possible trees to verify this conjecture is not required. They show that searching in a small special family of trees known as caterpillars (a tree is a caterpillar if the deletion of all its end vertices produces a path) suffices and hence achieving the first polynomial time algorithm up to integer n to verify the conjecture. They also provide many efficient algorithms for computing trees with given Wiener indices, and implementation results show that their performance is asymptotically better than their theoretical worst-case upper bound. The authors also point out that the approaches in their paper can be used as general techniques for tree construction problems in combinatorial biology and chemistry when it is necessary to deal with tree classes.

In [24], a sharp upper bound for $\mu(G)$ of a graph is given depending on the order of a graph and the independence number (maximum size of an inde-

pendent (pair-wise nonadjacent) set of vertices of G). The author answers the question posed by Erdos asking for bounds on the independence number of a graph with a given $\mu(G)$. The author also gives the upper and lower bounds on $\mu(G)$ depending on the matching number (maximum size of an independent (pair-wise nonadjacent) set of edges of G).

Dankelmann [23], gives an algorithm for computing $\mu(G)$ on an interval graph of size $o(n^2)$ in time $O(m)$ (recall that m denote the number of edges in a graph). This implies that for such a graph and size, $W(G)$ can be computed in time $O(m)$. In [23], apart from computing $\mu(G)$ of an interval graph, it is also argued that when G is a tree instead of interval graphs, $\mu(G)$ of a tree of order n can be computed in time $O(n)$ which is optimal. Thus when G is a tree, $W(G)$ can be computed in time $O(n)$.

Iyer and Reddy [85] obtained a closed-form expression for Wiener index of binomial tree. They also provide efficient algorithms for computing the Wiener indices of Fibonacci trees and binary Fibonacci trees with F_k (k-th Fibonacci number) vertices in time $O(\log F_k)$. This bound is an improvement over the bound obtained in [23]. That is, in [23] for any tree T with n vertices, $W(T)$ can be computed by an algorithm in time $O(n)$.

Although algorithms are available for average distance on strongly chordal graphs based on the APSP problem which runs in time $O(n^2)$, the author in [83] obtained a new algorithm that is not dependent on the APSP problem for average distance on strongly chordal graphs which runs in $O(n^2)$. Though the time bound is same, but the algorithm in [83] runs better than the previous algorithms on an average.

Very recently S. Klavžar *at el.* [55], proposed the average distance in interconnection networks via reduction theorems for vertex-weighted graphs. Their idea is to first shrink the original graph into smaller weighted graphs called quotient graphs. Then shrink this quotient graphs further into smaller weighted graphs called reduced graphs. During this shrinking process, a part of the Wiener index of the bigger graph is added as a corresponding weight to the smaller graph. Finally, the Wiener index of the original graph is calculated by the way of the Wiener index of the weighted reduced graphs. They have also demonstrated the significance of this technique by computing the average distance of butterfly and hypertree architectures. For other results on the average distance see [55] and references therein.

3.3 Concluding remarks and open issues of Wiener index or average distance

The problem of finding Wiener index and average distance of a graph are studied extensively in the literature. For general graphs, obtaining asymptotically faster algorithm for Wiener index (or average distance) directly without the application of APSD problem of a graph was not so obvious. So, researchers have explored on restricted family of graphs to obtain a better algorithm for computing Wiener index (or average distance) than the APSD problem. Based on the results aforementioned, we conclude this section with few open issues and future research in algorithmic computation of Wiener index (or average distance) of a graph:

- Is there a genuinely asymptotically faster algorithm for computing average distance in general than computing all distances between vertices of the graph [15]?

- Dankelmann [23] gave an algorithm for computing $\mu(G)$ of an interval graph G of size $o(n^2)$ in time $O(m)$. Thus it would be interesting to know for which other graph classes $\mu(G)$ can be computed in $O(m)$ time. One of the possible candidate could be strongly chordal graphs. In [83], an algorithm for $\mu(G)$ on strongly chordal graphs obtained $O(n^2)$ time bound. An interesting problem is, whether a similar technique can be extended to obtain that runs in linear time in the size of input graph.

- In [85], the results on Fibonacci trees and binary Fibonacci trees with F_k (k-th Fibonacci number) vertices, algorithms are obtained for computing their Wiener indices in time $O(\log F_k)$. For definitions of Fibonacci trees and binary Fibonacci trees and their applications we refer [85]. It would be desirable to obtain a closed form expression for computing their Wiener indices.

- Zmazek and Žerovnik [89] gave a linear time algorithm for weighted Wiener index on weighted cactus graphs (a graph is cactus if every edge lies on at most one cycle). The authors in [55], gave the result by introducing a new technique for computing the weighted Wiener index of butterfly networks and hypertree networks. But their method is applicable only to those families of graphs that partitions the edge set of a network into components using transitive closure which in turn enables their weighted Wiener index to compute efficiently. Thus it would be interesting to determine a unified way that is applicable on different graph

classes. In addition, using the newly introduced technique in [55], the open issue is, the computation of other topological indices.

Finally we consider the minimum average distance spanning tree problem.

4 MAD trees

In addition to average distance of a network, suppose if one is interested in designing a tree subnetwork of a given network such that the delay in sending a message between any two nodes of the network is minimum on the average, then such a tree can be modeled by finding the minimum average distance spanning tree (MAD) trees (MAD trees are also known as minimum routing cost spanning trees) [22]. Thus MAD trees also play an important role in communication networks [52]. It is well known that finding a minimum-cost spanning tree is one of the classic problem in algorithmic graph theory. Also there is an interest in finding the best spanning tree with important parameters such as minimum radius, minimum diameter, and minimum average distance (see [21]). In this section the following problem is considered: Statement of the problem: Given a connected, undirected graph G, the goal is to simply find a spanning tree of G whose average distance is minimum.

4.1 Computation of MAD trees

The problem of finding a MAD tree in general is NP-hard [52]. So the natural question that arises is, for which special graph classes a MAD tree can be found in polynomial time. As mentioned in [21], Entringer et al. [36] showed that there is a spanning tree whose average distance is less than twice the average distance of the original, and that such a tree can be found in polynomial time. Later, a polynomial time approximation scheme for minimum routing cost spanning trees has been developed by Wu et al. [86]. For further results on MAD trees can be found in [35, 36]. In [21], an linear time algorithm is exhibited for computing a MAD tree of a given distance-hereditary graph. In [22] the average distance $\mu(G, w)$ of vertex-weighted graph (G, w) is defined as follows. If G is a graph with weight function $w : V(G) \to \mathbb{R}^+$, then

$$\mu(G, w) = \binom{w(V)}{2}^{-1} \sum_{u \in V(G)} \sum_{v \in V(G)} w(u)w(v)d(u, v),$$

where $w(V)$ is the total weight of the vertices in G. Dahlhaus et al. [22], show that for a given interval graph G a MAD tree can be computed in time $O(m)$.

If an interval representation of an interval graph G with n vertices is given and the left and the right boundaries of intervals are sorted, then a MAD tree of G can be computed in $O(n)$ time. Dobrynin *et al.* [31] conjectured the following problem: the binomial tree is a MAD tree of the hypercube H_k. Tchuente *at el.* [79] made a step towards the proof of this conjecture by showing that the binomial tree B_k is a local optimum with respect to the 1-move heuristic (A 1-move consists of adding to a tree H, an edge e and removing an edge e' from the unique cycle created by e).

Recently, Jana and Mondal [50], and Mondal [60], obtained efficient algorithms for computing MAD tree on permutation graphs and trapezoid graphs respectively in $O(n^2)$ time bound based on breadth-first tree.

4.2 Concluding remarks and open issues of MAD trees

Although the problem of finding a MAD tree is NP-hard in general, but fortunately there is a possibility of obtaining a polynomial time algorithm for restricted family of graphs. Based on the results aforementioned, we conclude this section with few open problems and future research in computing a MAD tree:

- Is there a polynomial time algorithm to find a MAD tree of a vertex weighted interval graph [21]? The authors in [21] point out that using their existing technique, the possible candidates for future research could be strongly chordal graphs.
- It would also be interesting to know for which other restricted graph classes a MAD tree can be computed in polynomial time. Because the result on distance-hereditary graphs is known in polynomial time, so apart from the strongly chordal graphs, the other possible candidate could also be chordal bipartite graphs.
- Hypercube is a well known and popular interconnection network for multicomputers. The question whether the binomial tree is a MAD tree of the hypercube remains open (see [31]) even though Tchuente *at el.* [79] made a step towards this open problem.

5 Conclusions

In this paper, we reviewed studies on the all-pairs shortest paths problem and its variants namely, the Wiener index, the average distance, and the minimum

average distance spanning tree (MAD tree) problem in graphs. Each of these problems are undoubtedly fundamental and classical problems in graph algorithms and is well known in the research community. The significance of such problems are rich in applications. From the perspective of all-pairs shortest paths problem, the discussions are focused on the exact results of the static version. From the perspective of Wiener index, average distance, and MAD trees, the discussions are focused mostly on the algorithmic computation of such problems. Finally, under each section, some of the major open issues and future research are discussed on algorithmic determination and obtaining closed form expressions for each of such problems.

References

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesely, Reading, 1974. ⇒19, 22

[2] A. Alon, Z. Galil, O. Margalit, go , *J. Comput. Sys. Sci* **54,** 2 (1997) 255–262. ⇒23, 25

[3] M. J. Atallah, D. Z. Chen, D. T. Lee, An optimal algorithm for shortest paths on weighted interval and circular arc graphs, with applications, *Proc. European Sympos. on Algorithms*, *Lecture Notes in Computer Science* **726** (1993) 13–24. ⇒20

[4] A. T. Balaban, Applications of graph theory in chemistry, *J. Chem. Inf. Comput, Sci.* **25,** 3 (1985) 334–343. ⇒28

[5] V. Balachandhran, C. Pandu Rangan, All-pairs-shortest length on strongly chordal graphs, *Discrete Appl. Math.* **69,** 1–2 (1996) 169–182. ⇒20

[6] Y. A. Ban, S. Bereg, N. H. Mustafa, A conjecture on Wiener indices in combinatorial chemistry, *Algorithmica* **40,** 2 (2004) 99–117. ⇒17, 30

[7] G. E. Blelloch, V. Vassilevska, R. Williams, A new combinatorial approach for sparse graph problems, *Proc. International Colloquium on Automata, Languages and Programming* **35** (2008) 108–120. ⇒25

[8] A. Brandstädt, V. B. Le, J. P. Spinrad, *Graph Classes: A Survey*, *SIAM Monogr. Disc. Math. Appl.* 1999. ⇒27

[9] R. M. Casablanca, Average distance in the strong product of graphs, *Util. Math.* **94** (2014) 31–48. ⇒29

[10] T. M. Chan, All-pairs shortest paths with real weights in $O(n^3/\log n)$ time, *Proc. 9th Workshop Algorithms Data Struct.*, *Lecture Notes in Computer Science* **3608** (2005) 318–324. Also available in *Algorithmica* **50,** 2 (2008) 236–243. ⇒22

[11] T. M. Chan, All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time, *Proc. ACM-SIAM Sympos. Discrete Algorithms* **17** (2006) 514–523. ⇒25

[12] T. M. Chan, More algorithms for all-pairs shortest paths in weighted graphs, In *Proc. 39th ACM Symposium on Theory of Computing (STOC)* (2007) 590–598. ⇒22

[13] V. Chepoi, S. Klavžar, The Wiener index and the Szeged index of benzenoid systems in linear time, *J. Chem. Inf. Comput. Sci.* **37,** 4 (1997) 752–755. ⇒29
[14] V. Chepoi, S. Klavžar, Distances in benzenoid systems: further developments, *Discrete Math.* **192,** 1–3 (1998) 27–39. ⇒29
[15] F. R. K. Chung, The average distance and the independence number, *J. Graph Theory* **12,** 2 (1988) 229–235. ⇒29, 32
[16] E. Cohen, Using selective path-doubling for parallel shortest-path computations, *J. Algorithms* **22,** 1 (1997) 30–56. ⇒20
[17] E. Cohen, Polylog-time and near-linear work approximation scheme for undirected shortest paths, *J. ACM* **47,** 1 (2000) 132–166. ⇒20
[18] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, *J. of Symb. Comput.* **9,** 3 (1990) 251–280. ⇒23
[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms* (3rd edition), The MIT Press, 2009. ⇒16, 17, 21, 24, 27
[20] E. Dahlhaus, Optimal (parallel) algorithms for the all-to-all vertices distance problem for certain graph classes, *Proc. of the International Workshop Graph-Theoretic Concepts in Comput. Sci.*, *Lecture Notes in Computer Science* **657** (1992) 60–69. ⇒20
[21] E. Dahlhaus, P. Dankelmann, W. Goddard, H. C. Swart, MAD trees and distance-hereditary graphs, *Discrete Appl. Math.* **131,** 1 (2003) 151–167. ⇒ 33, 34
[22] E. Dahlhaus, P. Dankelmann, R. Ravi, A linear-time algorithm to compute a MAD tree of an interval graph, *Inf. Process. Lett.* **89,** 5 (2004) 255–259. ⇒33
[23] P. Dankelmann, Computing the average distance of an interval graph, *Inform. Process. Lett.* **48,** 6 (1993) 311–314. ⇒29, 31, 32
[24] P. Dankelmann, Average distance and independence number, *Discrete Appl. Math.* **51,** 1–2 (1994) 75–83. ⇒29, 30
[25] P. Dankelmann, Average distance and domination number, *Discrete Appl. Math.* **80,** 1 (1997) 21–35. ⇒29
[26] P. Dankelmann, Average distance in weighted graphs, *Discrete Math.* **312,** 1 (2012) 12–20. ⇒29
[27] E. Dijkstra, A note on two problems in connexion with graphs, *Numerische mathematik* **1,** 1 (1959) 269–271. ⇒21
[28] C. Demetrescu, G. F. Italiano, Fully dynamic transitive closure: breaking through the $O(n^2)$ barrier, *Proc. IEEE Sympos. on Found. Comput. Sci.* **41** (2000) 381–389. ⇒20
[29] J. K. Doyle, J. E. Graver, Mean distance in a graph, *Discrete Math.* **17,** 2 (1977) 147–154. ⇒29
[30] W. Dobosiewicz, A more efficient algorithm for the min-plus multiplication, *Int. J. Comput. Math.* **32,** 1–2 (1990) 49–60. ⇒22

[31] A. A. Dobrynin, R. Entringer, I. Gutman, Wiener index of trees: theory and applications, *Acta Appl. Math.* **66,** 3 (2001) 211–249. ⇒17, 28, 29, 30, 34

[32] A. A. Dobrynin, I. Gutman, S. Klavžar, P. Žigert, Wiener index of hexagonal systems, *Acta Appl. Math.* **72,** 3 (2002) 247–294. ⇒30

[33] F. F. Dragan. Estimating all pairs shortest paths in restricted graph families: a unified approach. *J. of Algorithms* **57,** 1 (2005) 1–21. ⇒17, 26, 27

[34] R. C.Entringer, D. E. Jackson, D. A. Synder, Distance in graphs, *Czech. Math. J.* **26,** 2 (1976) 283–296. ⇒30

[35] R. C. Entringer, Distance in graphs: trees, *J. Combin. Math. Combin. Comput.* **24** (1997) 65–84. ⇒33

[36] R. C. Entringer, D. J. Kleitman, L. A. Sžekely, A note on spanning trees with minimum average distance, *Bull. Inst. Combin. Appl.* **17** (1996) 71–78. ⇒33

[37] A. X. Falcao, J. K. Udupa, S. Samarasekera, S. Sharma, B. E. Hirsch, R. A. Lotufo, User-steered image segmentation paradigms: Live wire and live lane, *Graphical Models and Image Process.* **60,** 4 (1998) 233–260. ⇒16

[38] T. Feder, R. Motwani, Clique patritions, graph compression and speeding-up algorithms, *J. Comput. Sys. Sci*. **51,** 2 (1995) 261–272. ⇒25

[39] A. M. Fitch, and M. B. Jones, Shortest path analysis using partial correlations for classifying gene functions from gene expression data, *Bioinformatics* **25** (2009) 42–47. ⇒17

[40] M. Fredman, R. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. ACM* **34,** 3 (1987) 596–615. ⇒21, 24

[41] M. L. Fredman, New bounds on the complexity of the shortest path problem, *SIAM J. Comput*. **5,** 1 (1976) 49–60. ⇒22

[42] Z. Galil, O. Margalit, All pairs shortest distances for graphs with small integer length edges, *Inform. Comput.* **134,** 2 (1997) 103–139. ⇒23, 24, 25

[43] T. Hagerup, Improved shortest paths on the word RAM, *Proc. International Colloquium on Automata, Languages, and Programming (ICALP)*, *Lecture Notes in Computer Science* **1853** (2000) 61–72. ⇒24, 27

[44] K. Han, C. N. Sekharan, R. Sridhar, Unified all-pairs shortest path algorithms in the chordal hierarchy, *Discrete Appl. Math.* **77,** 1 (1997) 59–71. ⇒20, 26

[45] Y. Han, M. Thorup, Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space, *Symp. on Found. of Comput. Sci.* **43** (2002) 135–144. ⇒24

[46] Y. Han, Improved algorithm for all pairs shortest paths, *Inform. Process. Lett.* **91,** 5 (2004) 245–250. ⇒22

[47] Y. Han, An $O(n^3(\log\log n/\log n)5/4)$ time algorithm for all pairs shortest paths, *Proc. European Sympos. Algorithms*, *Lecture Notes in Computer Science* **4168** (2006) 411–417. ⇒22

[48] Y. Han, T. Takaoka, An $O(n^3 \log\log n/\log^2 n)$ time algorithm for all pairs shortest paths, *Proc. 13th SWAT*, *Lecture Notes in Computer Science* **7357** (2012) 131–141. ⇒23

[49] M. R. Henzinger, P. Klein, S. Rao, S. Subramanian, Faster shortest-path algorithms for planar graphs, *J. Comput. System Sci.* **55,** 1 (1997) 3–23. ⇒26

[50] B. Jana, S. Mondal, Computation of a minimum average distance tree on permutation graphs, *Annals of Pure and Appl. Math.* **2,** 1 (2012) 74–85. ⇒34
[51] D. Johnson, Efficient algorithms for shortest paths in sparse graphs, *J. ACM* **24,** 1 (1977) 1–13. ⇒21, 23, 24
[52] D. S. Johnson, J. K. Lenstra, A. H. G. Rinnooy-Kan, The complexity of the network design problem, *Networks* **8,** 4 (1978) 279–285. ⇒33
[53] V. King, Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs, *Proc. IEEE Sympos. on Found. of Comput. Sci.* **40** (1999) 81–89. ⇒20
[54] S. Klavžar, and I. Gutman, Wiener number of vertex-weighted graphs and a chemical applications, *Discrete Appl. Math.* **80,** 1 (1997) 73–81. ⇒18
[55] S. Klavžar, P. Manuel, M. J. Nadjafi-Arani, R. Sundara Rajan, C. Grigorious, S. Stephen, Average distance in interconnection networks via reduction theorems for vertex-weighted graphs, *To appear* ⇒28, 29, 31, 32, 33
[56] F. Le Gall, Faster algorithms for rectangular matrix multiplication, *Proc. 53rd FOCS* (2012) 514–523. ⇒23, 25, 27
[57] P. Mirchandani, A simple $O(n^2)$ algorithm for the all-pairs shortest path problem on an interval graph, *Networks* **27,** 3 (1996) 215–217. ⇒20
[58] B. Mohar, T. Pisanski, How to compute the Wiener index of a graph, *J. Math. Chem.* **2** (1988) 267–277. ⇒29
[59] S. Mondal, M. Pal, T. K. Pal, An optimal algorithm for solving all-pairs shortest paths on trapezoid graphs, *Int. J. Comp. Eng. Sci.* **3,** 2 (2002) 103–116. ⇒26
[60] S. Mondal, An efficient algorithm for computation of a minimum average distance tree on trapezoid graphs, *J. Scientific Research and Reports* **2,** 2 (2013) 598–611. ⇒34
[61] E. N. Mortensen, W. A. Barrett, Interactive segmentation with intelligent scissors, *Graphical Models and Image Process.* **60,** 5 (1998) 349–384. ⇒16
[62] S. Mukwembi, Average distance, independence number, and spanning trees, *J. Graph Theory* **76,** 3 (2014) 194–199. ⇒29
[63] W. Peng, X. Hu, F. Zhao, J. Su, A fast algorithm to find all-pairs shortest paths in complex networks, *Procedia Comp. Sci.* **9** (2012) 557–566. ⇒16
[64] S. Pettie, A new approach to all-pairs shortest paths on real-weighted graphs, *Theoret. Comput. Sci*. **312,** 1 (2004) 47–74. ⇒21, 24
[65] S. Pettie, V. Ramachandran, A shortest path algorithm for real-weighted undirected graphs, *SIAM J. Comput.* **34,** 6 (2005) 1398–1431. ⇒24
[66] S. Peyer, D. Rautenbach, J. Vygen, A generalization of Dijkstras shortest path algorithm with applications to VLSI routing, *J. Discrete Algorithms* **7,** 4 (2009) 377–390. ⇒16
[67] M. Randic, Chemical graph theory-facts and fiction, *Ind. J. Chem.* **42A** (2003) 1207–1218. ⇒28
[68] R. Ravi, M. V. Marathe, C. Pandu Rangan, An optimal algorithm to solve the all-pair shortest path problem on interval graphs, *Networks* **22,** 1 (1992) 21–35. ⇒20

[69] A. Saha, M. Pal, T. K. Pal, An optimal parallel algorithm for solving all-pairs shortest paths problem on circular-arc graphs, *J. Appl. Math. and Computing* **17** 1 (2005) 1–23. ⇒27

[70] J. P. Schmidt, All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings, *SIAM J. Comput.* **27,** 4 (1998) 972–992. ⇒16

[71] R. Seidel, On the all-pairs shortest path problem in unweighted undirected graphs, *J. Comput. Sys. Sci.* **51,** 3 (1995) 400–403. ⇒23, 24, 25

[72] T. Shinn, T. Takaoka, Combining all pairs shortest paths and all pairs bottleneck paths problems, *Lecture Notes in Computer Science* **8392** (2014) 226–237. ⇒23

[73] A. Shoshan, U. Zwick, All pairs shortest paths in undirected graphs with integer weights, *Proc. IEEE Sympos. on Found. of Comput. Sci.* **40** (1999) 605–614. ⇒23, 27

[74] C. Sommer, *Approximate shortest path and distance queries in networks*, PhD thesis, The University of Tokyo, 2010. ⇒20

[75] R. Sridhar, D. Joshi, N. Chandrasekharan, Efficient algorithms for shortest distance queries on interval, directed path and circular arc graphs, *Proc. Int'l. Conf. on Comput. and Inf.* **5** (1993) 31–35. ⇒20

[76] V. Strassen, Gaussian elimination is not optimal, *Numer. Math.* **13,** 4 (1969) 354–356. ⇒23

[77] T. Takaoka, A new upper bound on the complexity of the all pairs shortest path problem, *Inform. Process. Lett.* **43,** 4 (1992) 195–199. ⇒22

[78] T. Takaoka, A faster algorithm for the all-pairs shortest path problem and its application, *Proc. 10th Int. Conf. Comput. Comb.*, *Lecture Notes in Computer Science* **3106** (2004) 278–289. ⇒22

[79] M. Tchuente, P. M. Yonta, J. Nlong II, Y. Denneulin, On the minimum average distance spanning tree of the hypercube, *Acta Appl. Math.* **102,** 2–3 (2008) 219–236. ⇒34

[80] M. Thorup, Undirected single-source shortest paths with positive integer weights in linear time, *J. ACM* **46,** 3 (1999) 362–394. ⇒24

[81] M. Thorup, Integer priority queues with decrease key in constant time and the single source shortest paths problem, *J. Comp. Sys. Sci.* **69,** 3 (2004) 330–353. ⇒24

[82] N. Trinajstic, *Chemical Graph Theory*, CRC Press, Boca raton, FL, 1992. ⇒28

[83] K. R. Udaya Kumar Reddy, Computing average distance on strongly chordal graphs, *National J. Tech.* **8,** 1 (2012) 26–35. ⇒31, 32

[84] V. Vassilevska Williams, Multiplying matrices faster than Coppersmith-Winograd, *Proc. 44th ACM Symposium on Theory of Computing* (to appear, 2012). ⇒23

[85] K. V. Iyer, K. R. Udaya Kumar Reddy, Weiner index of binomial trees and Fibonacci trees, *Int. J. Math. Engg. with Comp.* **1** (2010) 27–34. Also available at http://arxiv.org/abs/0910.4432. ⇒31, 32

[86] B. Y. Wu, G. Lancia, V. Bafna, K. M. Chao, R. Ravi, C. Y. Tang, A polynomial-time approximation scheme for minimum routing cost spanning trees, *SIAM J. Comput.* **29,** 3 (2000) 761–778. ⇒33

[87] L. Xu, X. Guo, Catacondensed hexagonal systems with large Wiener numbers, *MATCH Commun. Math. Comput. Chem.* **55,** 1 (2006) 137–158. ⇒28

[88] S. J. Xu, R. Gysel, D. Gusfield, Minimum average distance clique trees, *SIAM J. Discrete Math.* **29,** 3 (2015) 1706–1734. ⇒28, 29

[89] B. Zmazek, J. Žerovnik, Computing the weighted Wiener and Szeged number on weighted cactus graphs in linear time, *Croat. Chem. Acta.* **76,** 2 (2003) 137–143. ⇒32

[90] U. Zwick, Exact and approximate distances in graphs: a survey, *Proc. European Symp. on Algorithms* **9** (2001) 33–48. ⇒18, 27

[91] U. Zwick, All-pairs shortest paths using bridging sets and rectangular matrix multiplication, *J. ACM* **49,** 3 (2002) 289-317. ⇒22, 23, 25

[92] U. Zwick, A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths, *Proc. Int. Sympos. Algorithms and Computation*, *Lecture Notes in Computer Science* **3341** (2004) 921–932. ⇒22

Received: February 13, 2016 • Revised: March 18, 2016

DE GRUYTER OPEN
Acta Univ. Sapientiae, Informatica **8,** 1 (2016) 41–62

DOI: 10.1515/ausi-2016-0003

Contralog: a Prolog conform forward-chaining environment and its application for dynamic programming and natural language parsing

Imre KILIÁN
University of Dunaújváros
Dunaújváros, Hungary
email: kilian.imre@uniduna.hu

Abstract. The backward-chaining inference strategy of Prolog is inefficient for a number of problems. The article proposes Contralog: a Prolog-conform, forward-chaining language and an inference engine that is implemented as a preprocessor-compiler to Prolog. The target model is Prolog, which ensures mutual switching from Contralog to Prolog and back. The Contralog compiler is implemented using Prolog's de facto standardized macro expansion capability. The article goes into details regarding the target model.

We introduce first a simple application example for Contralog. Then the next section shows how a recursive definition of some problems is executed by their Contralog definition automatically in a dynamic programming way. Two examples, the well-known matrix chain multiplication problem and the Warshall algorithm are shown here. After this, the inferential target model of Prolog/Contralog programs is introduced, and the possibility for implementing the ReALIS natural language parsing technology is described relying heavily on Contralog's forward chaining inference engine. Finally the article also discusses some practical questions of Contralog program development.

Computing Classification System 1998: Computing methodologies–Logic programming and answer set programming
Mathematics Subject Classification 2010: 68T27
Key words and phrases: logic programming, mechanical theorem proving, forward chaining, dynamic programming, natural language parsing

1 Pro-Contra-Log: a two way street for inference

Robinson's resolution principle proved to be strong enough to build a whole language, and quite a programming school upon it. The backward chaining resolution strategy, together with the left to right and top down rule firing strategies have provided the Prolog language with well known, easily perceivable operational semantics, which in addition, resemble the traditional sequential languages, with the extra flavor of backtracking added [2] .

The other direction however, the forward chaining strategy has never opened such a clear way to follow, though several attempts were made. Though their motivations must have been quite different, spreadsheets, the event driven working mechanism of modern graphical user interfaces, and some CASE tools follow seemingly similar ways.

The most important reason for Prolog's success, as a programming language, could be that its resolution strategy is somehow an extension of the old-fashioned and well-known procedure-call semantics. At the same time, as a theorem prover, we consider it rather weak. Beside many factors, the deepest reason of this might be, that its strategy to manage implications contradicts common human sense. Instead of deducing new facts from existing facts in the direct way, Prolog tries to prove certain facts by applying implications in the inverse direction, i.e. from consequence to conditions.

On the other hand, similarly to the "divide et impera" strategy of algorithmic thinking, Prolog's strategy may also be inefficient. Proven facts are not stored; therefore if a similar fact should be proved later, then the inferential operations are performed again, sometimes causing thereby a significant loss of performance.

Since Prolog, as a language appeared and has proven its strength to certain problems, the need of integrating it with other programming languages and other software tools, was always a burning issue. Similarily, the integration with another logical programming language, i.e. with a language that implements another subset or another inference strategy for the same subset of logic, remained only a theoretical possibility.

The present article proposes a *programmer controlled tight integration* of the two approaches. Tight integration means that the two languages are syntactically conform, their target models are compatible, and their software packages can be integrated with each other. The integration is called programmer controlled, because by means of declarations and/or modularization the programmer is able and is supposed to influence and control the actual code generation strategy for the different code segments.

The programming languages in the present proposal are Prolog and its counterpart: Contralog. They are syntactically compatible, and their declarative semantics are the same. Their procedural semantics however, since they implement different resolution strategies, are different, and they also implement different non-logical controls. Since Contralog's target model is Prolog, a translator program is implemented, which compiles Contralog modules to Prolog. The connection between the two segments is implemented through exports and imports, and it is triggered in a programmer defined way. The common Prolog target model makes the transition between the two segments very easy, and it also allows the easy use of Prolog built-in predicates from Contralog clauses.

Contralog, as an extension of Prolog, maps Horn-clauses to a Prolog code so, that an incremental compiler transforms Contralog rules to Prolog. The resulting code thereby can be executed in a normal Prolog runtime environment. This also ensures that Contralog and Prolog codes can be mixed and invoked from each other. This composite system is called Pro-Contra-Log, or simply PC-Log.

2 The Contralog target model

In the Contralog runtime-model everything works in the inverse way than we are familiar with:

- *Inference starts* not by goals, but *by facts.*
- If there is a rule with a condition matching the new fact, then the rest of the condition literals are also checked. If we could already have proven the conditions earlier, the rule is told to *fire.* In such case the conclusion-fact is told to have been inferred.
- The term: *recently proven conclusion* means a new resolvent fact. This is stored in the blackboard (in dynamic Prolog clauses), and we always continue inferencing using this fact.
- Inference stops when *goal statements are reached.* These don't have any consequence side, thus inference operations need not be continued.
- Upon reaching a goal, or when by any reason, inference cannot be continued along the actual chain, the system *backtracks and searches for an earlier open alternative* in the inference chain.

The Prolog-Contralog transition can be activated in the following ways:

- In the precondition side of Contralog rules, literal "{}/1" results an *immediate call* to a Prolog goal.

- *Imports* in Contralog are those facts which are taken from somewhere else. Such facts start an inference chain, therefore Contralog imports are mapped to Prolog exports of the corresponding firing rules.

- On the other hand, Contralog *exports* are mapped to firing predicates of freshly deduced facts. Those predicates are either imported from another Contralog module, thus they are defined there, or simply are imported from another Prolog module of the runtime environment. Contralog exports become Prolog imports (even though Prolog standard does not know this lingual construct).

The basic problem in forward chaining inference is that a rule might refer to more than a single condition. In case when not all of them can be satisfied, we must wait until the rest becomes true, and the rule can be fired only afterwards. We are solving this by storing the inferred facts in dynamic predicates. Furthermore, for each Contralog condition literal we construct a Prolog rule that checks if the other preconditions are already satisfied.

Let us regard the following Contralog rule, as a simple example!

```
a:-b, c, d.
```

If b or c or d conditions have already been satisfied, then the resulting facts are stored in the corresponding b/0, or c/0 or d/0 dynamic predicates. Besides those, we map each precondition literal to a `fire_NAME` and a `test_NAME` Prolog predicate.

The `fire_FACT` Prolog calls test, whether the rest of conditions (each but NAME) have already been inferred. If they have, the rule triggers the evaluation of the consequence part.

The `store_FACT` Prolog calls serve to store the recently inferred fact, and finally they call the firing predicate. The actual implementation takes into account the declarations for fact withdrawal, and performs the necessary actions. For predicates blocking the resolution chain, the call of firing the rule is missing.

In the case above, the following Prolog code is constructed:

```
fire_b:- assert(b), test_b.
fire_c:- assert(c), test_c.
fire_d:- assert(d), test_d.
test_b:- c, d, fire_a.
test_c:- b, d, fire_a.
test_d:- b, c, fire_a.
```

2.1 Backtracking

As it was already hinted, this target model uses *backtracking strategy* for searching, like Prolog itself does. Upon entering a new Contralog fact, the inference process produces newer and newer consequence facts. During this process there are some *choice-points*. If the straight line of inference described above cannot be followed further, we say, the evaluation has reached a *dead end*. In case of a dead end, the inference engine looks back in the evaluation stack, searches for the last open choice-point, takes the next alternative solution, and continues the interpretation from this point.

The following cases may produce a choice-point:

- If a condition literal is referred to by more than a single Contralog rule, then a Prolog definition is constructed consisting of the same number of Prolog `fire_` rules. Their ordering to follow is the textual ordering.

- If a condition literal is satisfied several times, we store the same number of dynamic facts – provided they are not in the scope of the non logical declarations described later.

Trying new open choice points is called *backtracking*. Finding dead-ends in the inference chain may happen in several ways:

- If any condition does not hold in the given time. This may be either the failure of a Contralog condition, or, using the {}/1 construct, any immediate Prolog call.

- If, upon reaching a Contralog goal, we force the system to backtrack by any Prolog means.

2.2 Facts and goals

Facts and goals play somewhat an opposite role in Contralog, than in Prolog. Facts are basic information sources, which start the data driven resolution, therefore they are compiled to goals.

`fact.` a Contralog fact.

`:-fire_fact.` a Prolog goal.

Many Prolog systems do not handle the presence of several logical goals correctly, i.e. in case one of them fails, they do not call the next one. Because of this the actual implementation constructs a single predicate (*goal/0*) that invokes the firing predicate of each fact in the current module in an alternative sequence:

`fact1.` a Contralog fact1.

`fact2.` a Contralog fact2.

The example above generates the following predicate:

```
goal:-fire_fact1.
goal:-fire_fact2.
```

2.3 Interface elements

Similarly to the overall opposite nature of Contralog, interfaces in the compiled Prolog code also play an opposite role. For the sake of explanation the interface is predicate-based, and exports but also imports(!) are allowed.

A Contralog export means a consequence part of a clause, which is shared with the outside world. Therefore the corresponding firing Prolog call must be implemented outside of the module and is thus imported. For convenience the generated predicate name is slightly different from the firing predicate. Contralog imports mean some input of more basic data, which, seen from the view of the target model, would be again considered to be calls from outside, which transfer the information, and launch the corresponding forward chaining inference process. This means, the corresponding `fire_` auxiliary predicate is implemented inside, and thus must be exported.

2.4 Non-logical means to control inference

In order to control the overwhelming amount of consequence facts, the pure logic language must be provided with means to control. Similarly to Prolog, for the sake of practical programming, Contralog introduces some non-logical means to influence the resolutional strategy. The language therefore implements certain declarational forms for this purpose.

In each target module a predicate (`clean/0`) is generated that cleans the module-specific part of the blackboard (the dynamically generated facts) completely.

A predicate P is told to *block the inference chain*, if, whenever any successful application of resolution steps yields a fact p, matching P, then the fact p does not cause to search for matching conditions. That is, though its immediate consequences are not evaluated, the new fact still remains available for further resolution. In such a case the new fact is stored in the blackboard, but the corresponding firing predicate is not invoked. Such an operation is performed for predicates denoted by the `:-lazy NAME/ARITY.` declaration.

A predicate P is told to *withdraw its earlier results*, if, whenever any successful application of resolution steps yields a fact p, matching P, then some, or all of earlier resolved p1,...,pn facts, all matching P, are excluded from further resolution. In Contralog the following withdrawal schemes are implemented.

- *Full withdrawal.* The predicate P withdraws all of its earlier results. The behavior is similar to that of a normal global variable, that is upon inferring certain facts, other facts with the same signature are deleted from the blackboard. To reach such effect the declaration `:-var NAME/ARITY` can be used.

- *Key controlled withdrawal.* Certain arguments of the predicate are told keys. The predicate P withdraws only those earlier results, whose keys are matching those of the recently inferred fact, p. (The behavior is similar to that of a table in a relational database, where unique keys are defined, i.e. elements with same keys are simply overwritten.) This can be reached by the `:-key(NAME(KEYVECTOR))` declaration, where NAME is the name of the predicate, KEYVECTOR is a list of argument patterns. Pattern "+" in KEYVECTOR denotes that the argument belongs to a (composite) key, pattern "–" denotes the opposite.

3 Contralog programming examples

3.1 Pythagoras' triads

Generating Pythagoras' triads can be among the first examples of beginners' Prolog courses. Let's see, how it works with Contralog. Peano's axiom is expressed by a predicate of two clauses, similarly to Prolog:

```
natural(1).
natural(X):- natural(X1), {X is X1+1}.
```

A Contralog recursion may also start an endless inference chain: stating the first fact launches the rule application that launches it again and again. This can be avoided by placing the recursive clause as the last among the referring clauses. This means when `natural/1` fires, each other firing procedure attached to `natural/1` is called before the rule above. Among these, those procedures which generate Pythagoras' triads are also fired. Firing `natural/1` for its own recursive rule occurs only when the generated triads do not satisfy Pythagoras' condition, or, when upon finding a perfect triad, the user asks for a new result, thus forcing the system to backtrack.

One feasible strategy to generate integer pairs of a quarter-plane is to visit them in a diagonal way. For one diagonal, the sum of its x and y coordinates is invariant. (For convenience we allow here the X=0 value too.) The Contralog predicate, implementing this, is the following:

```
natural2(0,SUM,SUM):- natural(SUM).
natural2(X,Y,SUM):-
  natural2(X1,Y1,SUM),
  {X1<SUM, X is X1+1, Y is Y1-1, X<Y}.
```

Predicate `natural/1` produces the invariant sum of both (X and Y) coordinates. This, with X=0 value, also gives the first integer pair. The sum is passed on through the third parameter of `natural2/3`. Given the actual pair, the recursive second clause takes care to produce the next pair. Endless inference is blocked by referring to `natural2/3` in another predicate: it is also a precondition in predicate `natural3/4` that generates integer triads. On the other hand the arithmetical tests in `natural/2` ensure that numbers remain not only in the given quarter-plane (X1<SUM), but also in an eighth-plane (X<Y).

Visiting integer triads happens slightly differently. The last two (Y and Z) coordinates are generated by `natural2/3`, but the first coordinate (X) is

generated using the (0<X, X<Y) conditions, by a simple scanning of the (0;Y) integer range. Conditions (X>0,Y>0,Z>0) allow to search only in an eighth of the three-dimensional space, while (X<Y<Z) conditions part it to halves twice further on. A 32-fold reduction of the total searching space is a significant result in time, while we don't lose any characteristic results.

```
natural3(X,Y,Z,SUMXZ):-
  natural2(Y,Z,SUMXZ), {X is Y-1, X>0}.
natural3(X,Y,Z,SUMXZ):-
  natural3(X1,Y,Z,SUMXZ), {X is X1-1, X>0}.
```

Once we can generate integer triads, checking for Pythagoras' triads is an easy job by the following clause:

```
pyth3(X,Y,Z):-
  natural3(X,Y,Z,_),
  {SUM2 is X*X+Y*Y, SUM2 is Z*Z}.
```

There are several possibilities to make this program run:

- The most suitable way to start the program is to call `:-p3:goal.`, which is an automatically generated procedure in module `p3`. But if we leave the clause above in the present implicational form so, that no clause refers to `pyth3/3` in the condition side, then a reference is generated to its firing predicate, which remains unsatisfied, thus we get an undefined procedure Prolog error. If we declare `:-export([pyth3/3]).`, then, instead of a firing predicate, the Contralog translator generates a call to the Contralog-exported predicate `exp_pyth3/3`. For this we must provide a testing environment, a module to use our original module, and to define the missing predicate somehow like the following:

  ```
  exp_pyth3(X,Y,Z):-
    write([X-Y-Z]), nl.
  ```

- If we call thereby a simple goal, then it will generate triads, but for the first triad, matching Pythagoras' condition, the inference will stop. The generated results are displayed by the `exp_pyth3/3` predicate above. To get subsequent results the system must be forced to backtrack by the following way: `:-p3:goal, fail.`, or simply by pressing ";" in the SWI-Prolog environment.

- If we declare `:-lazy pyth/3.`, and we make the system backtrack, then it will generate Pythagoras' triads in and endless loop and it will collect the results in dynamic predicate `pyth3/3`, until some system limit is reached.

If we do all this, except the `lazy` declaration, then we get the first familiar result, and if – in the SWI-Prolog environment – we force the system to backtrack, than we may also get the rest of them.

```
?- p3:goal.
[3-4-5]
true ;
[6-8-10]
true ;
[5-12-13]
true ;
[9-12-15]
true
```

3.2 Dynamic programming: Optimization of matrix chain multiplication

Matrix chain multiplication is a typical example for dynamic programming [1]. Matrix multiplication is an associative, but not commutative operation. The actual parenthesization, however, influences the efficiency of the entire operation considerably.

When trying to find the optimal parenthesization, the classical "divide et impera" approach leads to an exponential time complexity, while the data driven dynamic solution remains polynomial. The reason of the combinatorial explosion is that "divide et impera" divides the problem two (or sometimes more) parts, which are presumed to be independent from each other. That is, a solution of one part cannot overlap with the solution of any other part. The approach may work even so, but in such cases it forgets the corresponding sub-results, and calculates them again and again multiple times. This causes the loss of efficiency.

The dynamic solution uses a triangle-matrix to store the intermediate results. One element, m(i,j) stores the optimal number of scalar multiplications necessary to compute the i to j (i<=j) subsection of the entire chain. For single element subchains the following supposition holds obviously.

$m(i, i) = 0$

For longer subchains (when j>i), the $\mathtt{m}(\mathtt{i},\mathtt{j})$ optimum value can be calculated by breaking the chain at index k (i<=k<=j), and reducing the problem to the optimums of the left part (m(i,k)) and the right part (m(k+1,j)). Optimizing means to find the k index when the derived sum is the cheapest. This is described by the following equation:

$$\mathtt{m}[\mathtt{i},\mathtt{j}] = \mathtt{min}_{\mathtt{i}<=\mathtt{k}<\mathtt{j}}(\mathtt{m}[\mathtt{i},\mathtt{k}] + \mathtt{m}[\mathtt{k}+1,\mathtt{j}] + \mathtt{row}(\mathtt{i}) * \mathtt{column}(\mathtt{k}) * \mathtt{column}(\mathtt{j}))$$

Here rows and columns denote the horizontal and vertical size of the i-th matrix in the chain. The algorithm uses another matrix $\mathtt{c}(\mathtt{i},\mathtt{j})$, to store the index of optimal parenthesization in the i to j subchain. This is also called cutting matrix.

If we try to program the recursive equation above by a recursive computer program, then the solution will follow the "divide et impera" principle. This is also the case for Prolog predicates. We have already hinted that the Contralog solution turns the original backward-chaining interpretation of Prolog to the opposite: data items, as already proven facts, are stored in dynamic predicates, and implications are interpreted in their natural way: from precondition to conclusion. We expect Contralog to allow the programs to be as simple, as the equation above, while it can also manage all the technical details of forward-chaining and/or dynamic programming.

The $\mathtt{m}(\mathtt{i},\mathtt{j})$ and $\mathtt{c}(\mathtt{i},\mathtt{j})$ matrices are stored in a single Contralog predicate `matrix(I,J,M,C)`. Parameters I and J are matrix indices, while M and C are the values of the optimum and cutting matrices.

At the start of the algorithm, the $\mathtt{m}(\mathtt{i},\mathtt{i}) = 0$ initializations are performed by the following Contralog code.

```
size(6).
matrix(SIZE,SIZE,0,0):-
  size(SIZ), {SIZE is SIZ-1}.
matrix(I,I,X,C):-
  matrix(I0,I0,X,C), {I is I0-1, I>=0}.
```

The role of asserting the fact `size(6)` is to launch an inference burst out. This, in the first step asserts only the `matrix(0,0,0,0)` fact, but in the following steps all consecutive $\mathtt{m}(\mathtt{i},\mathtt{i})$ values are generated in a cycle. Filling up the lowest and simplest layer (the $\mathtt{m}(\mathtt{i},\mathtt{i}) = 0$ values) itself is enough to start inference yielding the optimum for more complicated cases (longer matrix chains).

This is done by help of the following Contralog clause.

```
matrix(I,J,X,J1):-
  matrix(I,J1,Y,_), matrix(I1,J,Z,_),
  {I1 =:= J1+1, mxyz(I,I1,J,MULT),
  X is Y+Z+MULT,
    (matrix(I,J,X0,_)->X<X0;
    true)}.
```

The curly bracketed part of the clause is a direct Prolog call. We use this to perform simple arithmetic operations. The Prolog call `mxyz(I,K,J,M)` calculates the number of scalar multiplications necessary to multiply the result of the optimized (i,k) and (k+1,c) matrix subchain products so, that $M = \mathsf{row}(\mathsf{i}) * \mathsf{column}(\mathsf{k}) * \mathsf{column}(\mathsf{j})$ holds.

The predicate says: if there are two consecutive subchains, then a scalar multiplication number can be calculated for their concatenated chain. Then, according to the principle of gradual approximation, if we have found a value for the concatenated chain that is cheaper than we stored until now, then the more expensive value is replaced by the cheaper one.

The replacement of old optimum value with the new, cheaper one is solved by Contralog's `:-key(matrix(+,+,-,-)).` declaration. This declares the first two parameters (matrix indices) as keys, i.e. for a given (i,j) index pair only a single clause is allowed. The actual retracting of the old clause and asserting of the new one is done by the generated Prolog code in the background.

The Contralog program can be started by calling `:-goal.` The program will fail, which means there is no goal in the program, and there is no other, untried way for inference either. The result is the generated content of the `matrix/4` predicate. To display this, we must implement a simple cyclic procedure. Taking the example in [1], we get the following result. The first triangle is that of optimums, the latter is the cutting matrix. For a (30x35, 35x15, 15x5, 5x10, 10x20, 20x25) matrix chain, the total number of necessary multiplications can be read in the top-leftmost corner.

```
[15125,10500,5375,3500,5000,0]
[11875,7125,2500,1000,0]
[9375,4375,750,0]
[7875,2625,0]
[15750,0]
[0]
[2,2,2,4,4,0]
```

```
[2,2,2,3,0]
[2,2,2,0]
[0,1,0]
[0,0]
[0]
```

When examining the actual order of `matrix/4` facts, we can observe another interesting feature of Contralog. Namely: the inference is eager, or depth-first, like Prolog itself. An implication fires immediately as soon as the last precondition item has arrived. Contralog implements therefore a bottom-up (data-driven) and depth-first strategy for discovering the resolution graph of the problem. In our case this means that the lowest layer ($m(i,i)$ elements) could not have been generated yet, when the first inference steps are already done. This is because the first rule application to calculate the optimum for a chain of two matrices, (the 3-rd `matrix/4` clause) is performed as soon as the first two consecutive matrix-subchains ($m(5,5)$ and $m(4,4)$) are already generated.

3.3 Dynamic programming: Floyd-Warshall algorithm

Similarly to matrix chain multiplication, the problem of shortest paths in weighted graphs can also be solved quicker by dynamic programming approach than by recoursive programming. According to this principle we first give the recoursive definition of the problem, and then we create the sequential program to deliver partial results in a bottom-up manner. In the following example we shall show that if we transcribe the recoursive definition to Horn-clauses – or to Contralog, then Contralog's execution mechanism automatically generates the results of dynamic programming.

In the recoursive definition of the problem d_{ij}^{k} denotes the shortest path between vertices i-j so, that intermediate vertices can be chosen only from the first k vertices of the graph (the actual ordering of vertices is irrelevant). The trivial alternative of the recoursion is case $k = 0$, when no intermediate vertices may be used. In such a case the shortest path is equivalent to the corresponding i-j item of the graph's weight matrix. Otherwise the shortest path using at longest the first k vertices, is equal to the shortest path using k-1 vertices, or it contains vertex k.

$d_{ij}^{0} = w_{ij}$

$d_{ij}^{k} = min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$

The Horn-clause (Contralog) transcription of the definition above is the following:

```
wm(0,I,J,W,SIZE):- w(LL),
 {length(LL,SIZE),between(1,SIZE,I),
 nth1(I,LL,L), between(1,SIZE,J),
 nth1(J,L,W)}.
wm(K,I,J,W,S):- wm(K1,I,J,K1IJ,S),
 wm(K1,I,K,K1IK,S),wm(K1,K,J,K1KJ,S),
 {K is K1+1, K1=<S},
 {W is min(K1IJ, K1IK+K1KJ)}.
```

Parameters K,I and J in definition `wm/5` are obvious. Here, W means the functions value d_{ij}^{k}, while SIZE is the size of the graph (the number of its vertices). This parameter is passed along in order not be calculated it again and again at each step. Definition `wm/5` is the three dimensional matrix itself that consists of dynamic facts. Instead of a sequential cycle, the Contralog execution it is built up by the forward chaining inference mechanism.

The clue of the algorithm lies in the second statement. The three `wm/5` conditions refer to the three d_{ij}^{k} values in the arguments of `min` function. Subsequent Prolog calls are calculating the index-relations, they stop the cycle or perform the actual calculation of minimal value.

The Contralog program, introduced above, implements the evaluation strategy of dynamic programming perfectly with one difference. Namely: partial results are not produced layer-by-layer, not in a breadth-first, but rather in a depth-first way.

Definition `wm/1` contains the weight matrix in a single fact. The first statement of three `wm/5` produces the first element of the 0-th layer first, and upon backtracking it produces also the rest. After the production of each element the second statement of three `wm/5` also starts, and if there are elements in the previous layer for which the preconditions are met, then it also produces an element from the next layer. This may also produce an element of the second next layer and so on

Depth-first execution means, that one corner of `wm/5`, the three dimensional matrix is built up as high is possible, and only in case of no further steps, we backtrack and try to build the lower layers forward. But any element is inserted in a lower layer, building up the higher layers is immediately tried.

4 Applying Contralog for ReALIS natural language parsing

The basic principles of ReALIS (Reciprocal And Lifelong Interpretation System) research project, targeting natural language processing techniques, is described in many conference articles, and even in a book [6, 4]. From the aspect of a Contralog natural language parser, its most important principles are the following:

- *Total lexicality*: each kind of lingual information is stored in the lexicon (in the vocabulary) [5]. Lexical items basically follow the feature structured method, whereas each item may demand certain other lingual context, and it also may offer certain services. Parsing, according to this approach, means the exact discovering and unveiling the offer-demand relationship. This also means: there is no special repository for lingual rules; lexical items describe all relevant information to perform parsing and/or generating natural language texts.

- *Modal logical framework*: interpreters are modelled in the world, and worlds are modelled in the brain of interpreters. The world-model in the background is a hierarchical structure of world-contexts. The root-world corresponds to the objective, outside world, that contains objects, and also contains subjects, i.e. agents, being able to interpret sentences and to store their own world-model. The content of interpreters internal world-model may also contradict the root world model. On the other hand interpreters store an own internal world structure – different modal contexts are stored in a different worldlets (for seen, heard, read, believed information, etc.) The internal world model of interpreters is empty when born, and the content of their world model gradually increases during their life – not necessary monotonically, certain information pieces may also be mistaken, and can later be corrected or even erased.

- *Discourse representation theory*, integrated in the interpreter's world model. Parsing is not restricted to a single sentence, but is extended to all the sentences of a *discourse* (several sentences in the same thematic or situational context). These sentences are usually also in a certain relationship with each other (e.g. antecedent, consequent, argument, etc.). These are called *rhetoric relations*. In addition to simply taking over similar structures of earlier discourse-representation schools [7], we improve them slightly. First, they are not necessarily independent, so we

are using a minimal/canonical set of rhetoric relations. Second they are not necessarily objective hence they are not stored in the objective root world, but in the interpreter's subjective world model.

According to ReALIS, instead of calculating a parse tree, the primary goal of parsing is to produce the following four mathematical relations:

- α: the *entity anchoring relation* is an equivalence relation that connects equivalent entities in a discourse.

- κ: the *cursor cluster* that describes global contextual information (Here, Now, Ego, There, Then, You, etc.). Some of its elements (place, time, influence) act as cursors; i.e. in a real discourse they are automatically advanced sentence by sentence.

- λ: the *level relation* that relates rhetorical contexts and/or modal logic relationships [7]. The level structure shows a self-embedding nature, and it also specifies the availability scope of discourse object references.

- σ: the *eventuality relation*, that maps lexical items to logical expression fragments, and finally maps sentences and/or larger textual units to complete logical expressions.

Total lexicality means that beyond mere textual elements (words and/or morphemes), lexical items also contain their eventuality function (a logical expression), instructions for entity anchoring, and the rhetorical and/or modal anchoring instructions. Furthermore a very important piece of a lexical item is its *offer-demand relationship*. This is used to describe morphological and/or syntactical bindings, along with the strength and the direction of the bindings. According to these principles, parsing is nothing more than a sort of domino-game; each syntactical element has certain offers, and may demand certain other elements in the neighborhood. The calculation of the relations mentioned above is done by the underlying unification-based means of finding matching demands and offers.

The mentioned parsing strategy of ReALIS is suitable for any languages, but in practice we recommend it especially for languages, like Hungarian, with variable or free word order. The application of traditional parsing methods for free word order languages produce inefficient results because of the frequent need of backtracking.

4.1 Prolog target models

The most obvious and usual target model for designing Prolog programs is the *relational target model*. According to this, the program calculates the <input,output> relation, which, if we program carefully, enables calculating the relationship in both directions. That is, for a parsing project, the same program can calculate the parse tree for a sentence, and may also calculate the sentence for a given parse tree at the same time.

Relational target model, on the other hand, performs a depth-first search on the resolution graph that depicts the inference process. The efficiency of backtracking programs can be strongly reduced by the frequency and the depth of backtracking, and we guess, natural language parsing may well involve deep and frequent backtracking.

Instead of the relational target model we propose the *inferentional target model* for natural language parsing. Instead of the <input,output> relation, the inferentional model tries to calculate the input $\rightarrow$ output implication. That is, the program must be reformulated so that it can prove that output data in some sense is the logical consequence of input data.

Both for relational and inferentional target models a key expression is *non-determinism*. That means: Prolog programs in general may deliver more than a single equivalent result. The set of results are propagating further on, while other constraints in the calling programs may filter out certain non-applicable results, and in a fortunate situation the end-result is already definite.

Prolog programs, in general, implement a *deductive inference model*. Deductive inference means that, for a given set of basic facts, the set of logical consequences is calculated. From this point of view the direction of inference (forward or backward chaining) is completely irrelevant.

The other strategy is called *abductive inference model*. For abductive inference we are aware of the consequences and the basic rules for inference, and we are searching for the possible facts, based on which the consequences can be inferred.

Though Prolog programs are basically implementing deduction, with a very easy extension we can also apply them for abduction. In case of abduction usually not the entire set, but only a subset of possible facts is unknown. To implement abduction in Prolog, instead of programming fixed set of facts, an implementation of *backtrackable assert operation* is to be programmed, that asserts each possible value for a given fact in a backtrackable way, and at last it retracts the fact completely. This, for the first run asserts certain facts. These may be deleted (and/or reasserted with other values) upon backtracking. The

simplest implementation of backtrackable asserts is the following:

```
assertb(FACT):-
  (assert(FACT);
  retract(FACT), fail).
```

Speaking about inference, its *strategy* can be crucial. We have already mentioned: in a number of cases Prolog's *backward-chaining* is not efficient enough. In case of *forward-chaining*, inference rule application burst-outs (inference chains) are started in a data-driven way, upon the arrival of certain facts. They may however arrive at any time, in an asynchronous way; delayed or even in a changed order. One inference step is performed when each precondition holds, and the corresponding facts are accessible. Although it is possible to prune the branches of the inference tree, consequences are produced in their entire richness, but if any of them matches any goal, then the program stops.

One obvious advantage of forward chaining is that already proven facts are stored in the blackboard, and they may be referred later on, for any times.

4.2 Inferential target model for parsing

If we apply the inferential target model to lingual parsing, the input sentence must be stored in a series of facts. Program clauses can be derived from the offer-demand relation of lexical elements, and certain general goals are the constructs to stop (or to start?) the inference.

As a drawback, the model cannot be used for generating text.

When performing ReALIS parsing, it is practical to define the following cuts (layers) over the entire inference graph.

1. The *layer of morphological analysis*. The character stream on the input channel is packed to words by the lexical parser. The stream of words is parsed by a morphological parser. Although ReALIS also has a solution for morphology [5], for convenience we propose to use a commercial solution. The result of morphological parsing is a Contralog sequence of facts. To focus on syntactical parsing, in the following example we omit to deal with the problem of morphological analysis. Let's see the facts resulting from the following Hungarian sentence: "Petra vágyik arra a magas német úszóbajnokra" ("Petra desires-3SG that the tall German

swimming champion-SUB") [6].

```
word(petra,1,1,noun('Petra',proper,nom,sing-3)).
word(petra,1,2,verb('vágy',[],decl, pres, sing-3)).
word(petra,1,3,noun('az',pro(point),sub,sing-3)).
word(petra,1,4,art(def,cons)).
word(petra,1,5,adj('magas')).
word(petra,1,6,adj('német')).
word(petra,1,7,adj('úszó')).
word(petra,1,7,noun('bajnok',common,sub,sing-3)).
```

The arguments of the facts above are the following: 1. a discourse identifier 2. sentence index in the discourse 3. word index in the sentence 4. a Prolog structure describing the result of morphological analysis.

2. The *layer of grammatical dependency relations*. We are calculating the regent-argument (bidirectional) and adjunct-argument- (one directional) relations. The example below shows the regent-argument description of verb "vágyik" (desires). Hungarian verb "vágyik" demands a nominative argument (the subject), and a sublative argument (the object). The strength of the former binding is -7, of the latter is +7. The former annotation (-7) means, the subject may appear well before the verb in a loose distance. The other means the opposite: the object may appear after the verb, and arbitrary other words may appear between them. Both arguments should form a generalized quantifier determinant structure (proper noun, determinate article, adjective, etc.).

```
regArg2(ID,S,XV,verb('vágy',[],MODE,VTIME,AGR),
    XS,noun(SUBJ,SKIND,nom,AGR),-7,
    XO,noun(OBJ,OKIND,sub,OAGR),7):-
  verb(ID,S,XV,'vgy',[],MODE,VTIME,AGR),
  gqdet(ID,S,XS,SUBJ,SKIND,nom,AGR),order(XV,XS,-7,nei),
  gqdet(ID,S,XO,OBJ,OKIND,sub,OAGR),order(XV,XO,7,nei).
```

3. The layer of *eventuality relations*. In this layer the logical form of regent-adjunct structures is built up in a way, that their arguments logical form is supposed to be built up before. In the example below the predicate `regArg2` builds up the grammatical structure (the regent-argument relation), while `sigma3` prepares the overall logical expression as a result.

(`=../2` is a technical Prolog call that transforms time referent structure to grammatical time notation.)

```
sigma3(ID,S,XV,TIME,SUB,OB,CLAUSE):-
    regArg2(ID,S,XV,verb('vágy',[],MODE,VTIME,_AGR),
                  XS,SUBJ,_PRS,XO,OBJ,_PRO),
    TIME =.. [VTIME,_],
    sigma3(ID,S,XS,TIME,SUB,CLAUSE,
               (desire(TIME,SUB,OB):-CONS)),
    sigma3(ID,S,XO,TIME,OB,CONS).
```

As the logical form of the sentence above, we may get the following clause. (The double implication can be transformed easily into a conjunction on the precondition side)

```
CLAUSE=((desire(pres(T),SUB,OB) :- swim(T, OB),german(T, OB),
        tall(T, OB),champion(T, OB)) :- name(T,SUB,'Petra'))
```

4. The *layer of rhetorical and modal relations*. In this layer, rhetorical and modal logical relations are built up as described by the λ function. Although there have been theoretical investigations completed [8], up to the writing of the article we don't have any concrete results to demonstrate this.

5 Program development with Contralog

Contralog is available as a preprocessor for SWI-Prolog [9] that works on the basis of its macro extension mechanism (`term_expansion/2`) . This is unfortunately only de facto standard, therefore the seamless operation with other Prolog dialects is not guaranteed.

When developing Contralog programs, the Prolog module, defining the above mentioned macro expansion predicate, must be consulted first. (`clog.pl`). It is not enough only to load `clog.pl` in the application module, because in the time of reading the first (module head) declaration of the module, the Contralog pretranslator must already be active. The module declaration itself is handled otherwise by Prolog, even its export list is understood as Prolog exports.

To declare Contralog export, `:- export(EXPLIST).` declaration should be used.

In general, Contralog program clauses are to be placed in a Prolog program. To switch between normal Prolog clauses, and preprocessed Contralog clauses, the following two new directives can be used.

`:- contra.` that starts the processing of Contralog code

`:- pro.` that switches back to normal Prolog

Beside the Contralog to Prolog translator, at the moment there is no other development tool available. It is a bit clumsy, for example, to debug Contralog programs. Since there is no debugger, debugging is possible only by help of the Prolog debugger, and it may work only for those, who are familiar with the target model.

6 Summary and future work

We have defined Contralog, which is a Prolog-conform language, but instead of backward chaining, it uses forward-chaining inference. The language itself is built-up on the same syntax: Horn clauses, only its declaration forms are different from those in Prolog.

For this end we have developed a Prolog target model to perform forward chaining inference. This enables to translate Contralog clauses directly to Prolog, and execute them by the Prolog software environment.

The Contralog to Prolog translator has been implemented by using Prolog's macro extension mechanism. This approach, and the Prolog target model itself enables various possibilities to integrate forward and backward chaining inference in a programmer-controlled way.

To demonstrate these, the article introduces several examples: after the simplest examples two dynamic programming problem and ReALIS natural language parsing mechanism is described.

Future work may include extended Contralog program development possibilities and tools.

The natural language parsing mechanism has been tested only for a handful of sentences by translating the lingual information to Prolog (Contralog) manually. Any future work must aim to collect lingual information, the mechanical translation of these to the Contralog target model, and the development of the overall ReALIS parsing environment.

Acknowledgements

The author is grateful for the Hungarian Research Fund TÁMOP 4.2.2.C-11/1/KONV-2012-0005 (Well-being in the informational society) for the support of his research activities regarding natural language technologies, and also for the Hungarian Research Fund TÁMOP 4.2.2.D. to cover the Mathinfo2015 conference participation expenses. Special thanks to colleagues: Gábor Alberti and Márton Károly for their kind remarks and contribution.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms* (3rd edition), The MIT Press, 2009. ⇒50, 52

[2] W. F. Clockshin, C. S. Mellish, *Programming in Prolog*, Springer Verlag Berlin, Heiderlberg, New York, 1994. ⇒42

[3] G. Alberti, *ReALIS: Interpretators in the world, worlds in the interpretator* (in Hungarian: ReALIS. Interpretálók a világban, világok az interpretálóban). Akadémiai Kiadó, Budapest, 2011. ⇒55, 59

[4] G. Alberti, *ReALIS. An interpretation system which is reciprocal and lifelong.* Akadémiai Kiadó, Budapest, 2011. ⇒55

[5] G. Alberti, K. Balogh, J. Kleiber, A. Viszket, Total lexicalism and GASGrammars: A direct way to semantics *Proc. CICLing2003*, NLCS 2588, pp. 37–48, (ed Gelbukh, A.) Springer Verlag, Berlin 2003. ⇒55, 58

[6] G. Alberti, I. Kilián, Bipolar influence-chain families instead of lists of argument frames – the sigma function of ReALIS (In Hungarian: Vonzatkeretlisták helyett polaritásos hatáslánccsaládok) *Proc. MSzNyVII. Hungarian Conference of Computer Linguistics*, pp. 113–126, (ed: A. Tanács, V. Vincze) VII. Magyar Számítógépes Nyelvészeti Konferencia, MSzNy pp. 113-126, SzTE Informatikai Tanszékcsoport, Szeged. 2010. ⇒55, 59

[7] H. Kamp, J. van Genabith, U. Reyle, Discourse representation theory. *Handbook of Philosophical Logic*, Vol. 15., 125394. Springer Verlag, Berlin, 2011. ⇒55, 56

[8] M. Károly, Interpretation and modality - towards the implementation of ReALIS' λ-function. (In Hungarian: Interpretáció és modalitás avagy a ReALIS λ-fūggvényének implementációja felé.) (ed. V. Vincze, A. Tanács) VIII. Magyar Számítógépes Nyelvészeti Konferencia, MSzNy 284–296. SzTE Informatikai Tanszékcsoport, Szeged. 2011. ⇒60

[9] J. Wielemaker: An overview of the SWI-Prolog programming environment, *Proc. 13-th International Workshop on Logic Programming Environments*, ed: F. Mesnard, A. Serebenik, Katholieke Universiteit Leuven, Belgium, 2003. 1–16 pp. ⇒60

Received: February 9, 2016 • Revised: April 16, 2016

DE GRUYTER OPEN
Acta Univ. Sapientiae, Informatica **8,** 1 (2016) 63–81

DOI: 10.1515/ausi-2016-0004

Edge coloring of graphs, uses, limitation, complexity

Sándor SZABÓ
University of Pécs
Institute of Mathematics and Informatics, Ifjúság u. 6
7624 Pécs, HUNGARY
email: sszabo7@hotmail.com

Bogdán ZAVÁLNIJ
University of Pécs
Institute of Mathematics and Informatics, Ifjúság u. 6
7624 Pécs, HUNGARY
email: bogdan@ttk.pte.hu

Abstract. The known fact that coloring of the nodes of a graph improves the performance of practical clique search algorithm is the main motivation of this paper. We will describe a number of ways in which an edge coloring scheme proposed in [8] can be used in clique search. The edge coloring provides an upper bound for the clique number. This estimate has a limitation. It will be shown that the gap between the clique number and the upper bound can be arbitrarily large. Finally, it will be shown that to determine the optimal number of colors in an edge coloring is NP-hard.

1 Introduction

Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ be a graph. Here V is the set of nodes of the graph G and E is the set of edges of the graph. In this paper we will be dealing exclusively with finite graph, that is, it will be assumed that the sets V and E are finite. We further narrow the class of graphs we consider throughout this paper by

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 05C15
Key words and phrases: clique, maximum clique, maximal clique, clique search algorithms, vertex coloring, edge coloring.

excluding the graphs that have either loops or double edges. In other words we will be working with finite simple graph.

Let k be a fixed positive integer. A subgraph Δ of G is called a k-clique in G if each two distinct nodes of Δ are adjacent in G and Δ has k nodes. The number of edges in Δ is equal to $k(k-1)/2$. We refer to k as the size of the clique Δ. Sometimes we call Δ a clique of size k instead of a k-clique. A k-clique Δ in G is called a maximal clique if Δ is not a subgraph of any $(k+1)$-clique in G. A k-clique Δ in G is called a maximum clique if G does not contain any $(k+1)$-clique. A graph may have several maximum cliques. All maximum cliques in G have a well defined common size. This well defined number is referred as the clique number of G, and it is denoted by $\omega(G)$.

The expression "clique search problem" refers to a number of problems related to finding cliques in a given graph. One may look for maximal cliques or maximum cliques. One might be interested in listing all maximal cliques or listing all maximum cliques. We maybe content with locating only one maximum clique. Or we maybe satisfied with just learning the clique size of G without exhibiting any maximum clique. We describe some relevant clique search problems more formally.

Problem 1 *Given a finite simple graph* G *and given a positive integer* k. *Decide if* G *contains a* k*-clique.*

Problem 1 is a decision problem and it is well-known that it belongs to the NP-complete complexity class. (See [7].)

Problem 2 *Given a finite simple graph* G *and a positive integer* k. *List all* k*-cliques in* G.

Problem 2 is not a decision problem. It is clear that Problem 2 cannot be computationally less demanding than Problem 1.

Determining the clique number $\omega(G)$ of G is not a decision problem either. It is an optimization problem. But again it must be clear that finding $\omega(G)$ is computationally at least as challenging as Problem 1.

Clique search problems have many practical applications and there is a considerable amount of research devoted to them. For details see for example [1], [3], [4], [5]. Many practical clique search algorithms utilize the coloring the nodes of a graph. We color the nodes of a given finite simple graph G with k colors satisfying the following conditions.

(1) Each node receives exactly one of the colors.

(2) Adjacent nodes never receive the same color.

This is the most commonly encountered coloring of the nodes of a graph. It is referred as a legal or a well or a proper coloring of the nodes of G. In connection with each finite simple graph G there is a number of colors k such that the nodes of G have a legal coloring with k colors and the nodes of G do not have any legal coloring with $k-1$ colors. This well defined number k is called the chromatic number of G and it is denoted by $\chi(G)$. Coloring of graphs is a vast subject on its own. In this paper we take a rather narrow view of coloring. We are interested in coloring only from one reason. Coloring provides upper estimates for $\omega(G)$. Namely, $\omega(G) \leq \chi(G)$.

One can devise further coloring schemes to get new upper bounds for $\omega(G)$. For example we may color the edges of a graph G with k colors in the following way.

(1) Each edge receives exactly one color.

(2) If x, y, z are distinct nodes of a 3-clique in G, then the edges $\{x, y\}$, $\{y, z\}$, $\{x, z\}$ must receive three distinct colors.

(3) If x, y, u, v are distinct nodes of a 4-clique in G, then the edges $\{x, y\}$, $\{x, u\}$, $\{x, v\}$, $\{y, u\}$, $\{y, v\}$, $\{u, v\}$ must receive six distinct colors.

We call this type of coloring of the edges of G a legal or well or proper edge coloring.

For a given finite simple graph G there is a number of colors k such that the edges of G has a legal coloring with k colors and the edges of G does not admit any legal coloring using $k-1$ colors. This minimum number of colors is called the edge chromatic number of G and it is denoted by $\chi_{(e)}(G)$.

The reader can observe that the inequality

$$\omega(G)\,(\omega(G)-1)\,/2 \leq \chi_{(e)}(G)$$

must hold. So coloring the edges of a graph G can be used to establish an upper bound for $\omega(G)$.

The edge coloring scheme described here was proposed in [8]. It was mentioned that a large scale numerical experiment indicates that typically edge coloring provides better bounds for the clique number than the node coloring. On the hand the edge coloring is computationally more expensive than the node coloring. In Tables 1, 2, and 3 we presented some of the numerical results. These results were not reported earlier. The graphs we used are related

to the construction of certain codes. The captions of the tables simply refer to the related codes.

In the tables the columns labeled by $|V|$ and $|E|$ contain the numbers of the nodes and the edges of the graphs, respectively. Using the simplest sequential node and edge coloring procedures give the number of colors listed in the columns labeled by the words "node" and "edge", respectively. Finally, the column labeled by the word "estimate" lists the upper estimate of the clique size computed from the number of colors of the edges.

Let $G = (V, E)$ be a finite simple graph. Using G we construct a new auxiliary graph $\Gamma = (W, F)$. We set $W = E$ and the distinct edges $\{u, v\}$, $\{x, y\}$ of G will be adjacent nodes of the graph Γ if each of the unordered pairs

$$\{u, x\}, \quad \{u, y\}, \quad \{v, x\}, \quad \{v, y\}$$

is an edge of the graph G. The reader can verify that a legal coloring of the nodes of the auxiliary graph Γ corresponds to a legal coloring of the edges of the original graph G. We may refer to Γ as the derived graph of G.

2 Applications of edge coloring

In this section we discuss the relevance of edge coloring to clique search algorithms.

The edge coloring can be used as a preconditioning technique.

Suppose that the edge coloring of the graph $G = (V, E)$ is given by the function $f : E \to \{1, \ldots, t\}$. Here $f(\{x, y\}) = c$ means that the edge $\{x, y\}$ receives color c. The function f can be stored conveniently as a matrix M. The typical entry $m(x, y)$ of M is defined by

$$m(x, y) = \begin{cases} f(\{x, y\}), & \text{if} \quad \{x, y\} \in E, \\ 0, & \text{if} \quad \{x, y\} \notin E. \end{cases}$$

The number of the colors used for coloring the neighbors of the node x is equal to the number of the distinct colors appearing in row x of the matrix M. We may call this number the color degree of the node x in the graph G.

If x is a node of a k-clique Δ in G, then the color degree of the node x must be at least $k-1$. Therefore, if the color degree of the node x is less than $k-1$, then node x can be deleted from the graph G without loosing the clique Δ. In other words, when we are looking for a k-clique in the graph G we can delete safely each node whose color degree is less than or equal to $k-2$.

There is further way to exploit the edge coloring for preconditioning. The greedy sequential coloring of the edges of the given graph G as a side result provides us with the adjacency matrix or the linked lists representation of the derived graph Γ of G. An inspection can help to delete nodes or edges of Γ.

The edge coloring can be used as a pruning rule. One simply can add an edge colored graph to the Carraghan-Pardalos [2] clique search algorithm.

The Carraghan-Pardalos algorithm at a particular stage of its execution maintains two sets of nodes. The first one is the set of nodes U which consists of the nodes of an r-clique Δ. The second one is the set of nodes L which contains the nodes of G that have a chance to extend the clique Δ. We restrict the graph G to the set L. Let H be the graph spanned by the set L in G. The edges of H are colored as H inherits a coloring from G.

One can count the number of the distinct colors appearing as edge colors in H. Let this number be s. Using s one can estimate the clique size of H. Namely, if $\omega(H) = t$, then $t(t-1)/2 \leq s$ must hold. This means that $t \leq \left(1+\sqrt{1+8s}\right)/2$ must hold.

Suppose we are looking for a k-clique in the given graph G. It is clear that if $\omega(\Delta)+\omega(H) \leq k-1$, then choosing nodes from the set L the clique Δ cannot be extended to a k-clique. Thus, if

$$r + \left(1+\sqrt{1+8s}\right)/2 \leq k-1,$$

then at this node of the search tree one can terminate the search. In other words at this node we can prune the search tree. One can record the edge coloring of G using the matrix M described earlier. One can easily store two different edge colorings of G in the matrix M. Using two edge colorings enhances the efficiency of the pruning.

When the largest color class is very large compared with the others, then the edge coloring offers a new opportunity to estimate the clique size.

Let Δ be a maximum clique in the given graph G. Suppose that the color classes of the edges in G are $C_1, \ldots, C_k$ and $|C_1| \geq \cdots \geq |C_k|$ holds. If we delete the edges appearing in C_1, then we get a new graph G' from G. If Δ does not contain any edge from C_1, then Δ is a maximum clique in G' too. In this case $\omega(G) = \omega(G')$. If Δ does contain an edge from C_1, then it may happen that $\omega(G) = \omega(G') + 1$. In either case $\omega(G) \leq \omega(G') + 1$. Since G' has fewer edges than G computing or estimating $\omega(G')$ can be simpler than computing or estimating $\omega(G)$. In this way we may collect some information about the clique size of G.

The edge coloring can be used as a branching rule to devise a parallel clique search algorithm.

Let us assume that we are interested in deciding if the given graph G contains a k-clique. Here k is a given positive integer. Suppose that the edges of G are legally colored with t colors. Let $C_1, \dots, C_t$ be the colors classes of the edges such that $|C_1| \geq \cdots \geq |C_t|$. The edges of k-clique Δ in G can be colored with $r = (k(k-1))/2$ colors and cannot be colored with fewer colors. If $t < r$, then clearly G cannot contain any k-clique. For the remaining part of the argument we assume that $t \geq r$. Let

$$e_1 = \{x_1, y_1\}, \dots, e_s = \{x_s, y_s\} \tag{1}$$

be all the edges in the color classes $C_r, \dots, C_t$. Let G_i be the subgraph of G spanned by the set of nodes $N(x_i) \cap N(y_i)$ in G for each i, $1 \leq i \leq s$. Here $N(x)$ denotes the set of neighbors of the node x in the graph G. If G_i contains a $(k-2)$-clique for some i, $1 \leq i \leq s$, then G contains a k-clique. In this case our problem is solved.

For the remaining part we may assume that G_i does not contain any k-clique for each i, $1 \leq i \leq s$. It means that we can delete the edge $e_i = \{x_i, y_i\}$ form G without removing any k-clique from G. (When we delete the edge e_i we do not delete any of the nodes x_i and y_i.) Deleting the edges (1) from G we end up with a graph G' whose edges are legally colored with $r-1$ colors. Consequently, this graph G' cannot contain any k-clique.

The summary of our consideration is that locating a k-clique in G can be reduced to locating a $(k-2)$-clique in the graphs $G_1, \dots, G_s$. We replaced the original clique search problem by a large number of smaller clique search instances. These smaller problems can be attacked independently of each other and we can solve them using a number of processors simultaneously. In this sense the edge coloring can form the base of a parallel clique search algorithm.

When the number of the nodes is overly large, then we can divide the set of nodes of the graph into two disjoint sets. The clique sizes of the smaller subgraphs induced by these sets provide lower and upper bounds for the clique size of the original graph. Using edge coloring the upper bound can be improved.

Let G be a finite simple graph. We divide the set of nodes V into two disjoint subsets U and W. Let H, K be the subgraphs of G induced by the subsets U and W, respectively. We consider a bipartite subgraph L of G induced by the subsets U and W. Note that

$$\max\{\omega(H), \omega(K)\} \leq \omega(G) \leq \omega(H) + \omega(K)$$

holds. Setting $h = \omega(H)$, $k = \omega(K)$ the upper estimate is $\omega(G) \leq h + k$. Coloring the edges of the bipartite graph L provides a correction term r to modify the estimate to $\omega(G) \leq h + k - r$.

Suppose G contains a $(h+k)$-clique. In this case $\omega(G) = h+k$ and the bipartite graph L must contain a (h,k)-biclique. Note that this biclique has hk edges and the colors of these edges are pair-wise distinct. In other words, if the edges of L can be colored legally using less than hk colors, then the equation $\omega(G) = h+k$ cannot hold.

Suppose that the edges of L can be colored legally using s colors. If s is small compared to hk, then the upper estimate for $\omega(G)$ can be lowered. For the sake of definiteness let us assume that $h \geq k$. Choose an integer r such that

$$(k-r)h \leq s < (k-r+1)h.$$

It follows that

$$k - (s/h) \leq r < k - (s/h) + 1.$$

This r is the correction term to improve the estimate for $\omega(G)$.

The edge coloring can be used to construct cuts in the linear programming reformulation of the maximum clique problem.

The maximum clique problem can be reformulated in terms of a 0-1 linear program. Let $G = (V, E)$ be a finite simple graph with $V = \{v_1, \ldots, v_n\}$ and let Δ be a clique in G such that U is the set of nodes of Δ. To the clique Δ we assign an n-dimensional vector $\mathbf{x}^T = [x_1, \ldots, x_n]$ such that

$$x_i = \begin{cases} 1, & \text{if} \quad v_i \in U, \\ 0, & \text{if} \quad v_i \notin U. \end{cases}$$

We may call $\mathbf{x}$ the characteristic vector of the clique Δ.

We consider the following 0-1 linear program P. Maximize the objective function $x_1 + \cdots + x_n$ subject to the constraints $x_i + x_j \leq 1$, where the unordered pair $\{v_i, v_j\}$ is not an edge of G. Replacing the condition $x_i \in \{0, 1\}$ by $0 \leq x_i \leq 1$ for each i, $1 \leq i \leq n$ we get the real relaxation P′ of the 0-1 linear program P. An optimum solution of P′ provides an upper bound for the clique size $\omega(G)$ of G.

Feeding the program P′ into a linear program solver we typically get the optimum solution $[1/2, \ldots, 1/2]^T$ which leads to the estimate $\omega(G) \leq n/2$.

The polyhedron of the program P′ may have many vertices with non-integer components. There are inequalities in the form $a_1x_1 + \cdots + a_nx_n \leq b$ that slices down non-integer solutions from the polyhedron but not slicing down any integer solutions. Such inequalities are called cuts and they can be added to the constraints of the program P′ to improve the estimate for $\omega(G)$.

Suppose that the edges of G are legally colored using t colors and $C_1, \ldots, C_t$ are the color classes of the edges of G such that

$$e_1 = \{a_1, b_1\}, \ldots, e_s = \{a_s, b_s\}$$

are the edges in the color class C_i. Let $v_{i(1)}, \ldots, v_{i(r)}$ be all the distinct nodes of G among $a_1, b_1, \ldots, a_s, b_s$. Note that the set $\{v_{i(1)}, \ldots, v_{i(r)}\}$ cannot contain the nodes of any 3-clique in the graph G. Consequently, the inequality

$$x_{i(1)} + \cdots + x_{i(r)} \leq 2$$

can be added, as a cut, to the linear program P'.

3 The gap phenomenon

In 1955 J. Mycielski [6] has proved the next result.

Theorem 3 *For each positive integer* k *there is a graph* G *such that* $\chi(G) = k$ *and* G *does not contain any* 3*-clique.*

Since G does not contain any triangle, the edges of G have a proper coloring with exactly 1 color. On the other hand, as $\chi(G) = k$ the nodes of this graph do not have any legal coloring with $k + 1$ colors. In other words, the upper estimates for the clique size $\omega(G)$ provided by the coloring of edges of G coincides with $\omega(G)$ and the upper bound of the clique size provided by the coloring of the nodes can be arbitrarily large. This makes the edge coloring looking very good in comparison with the node coloring.

In this section we will construct a family of graphs for which the gap between the clique number and the edge chromatic number can be arbitrarily large.

Theorem 4 *Let us choose an integer* k *with* $k \geq 3$*. There is a graph* $L^{(k)}$ *such that* $\omega(L^{(k)}) \leq 4$ *and* $\chi_{(e)}(L^{(k)}) \geq k$*.*

Proof. Let $M^{(k)}$ be the Mycielski graph with parameter k. Let $u_1, \ldots, u_n$ be the nodes of $M^{(k)}$. The graph $L^{(k)}$ will have $2n$ nodes $x_1, \ldots, x_n, y_1, \ldots, y_n$. The unordered pair $\{x_i, y_i\}$ is an edge of $L^{(k)}$ for each i, $1 \leq i \leq n$. Further if the unordered pair $\{u_i, u_j\}$ is an edge of $M^{(k)}$, then we add the edges

$$\{x_i, x_j\}, \quad \{x_i, y_j\}, \quad \{y_i, x_j\}, \quad \{y_i, y_j\}$$

to $L^{(k)}$. Figure 1 shows the construction in the special case $k = 3$.

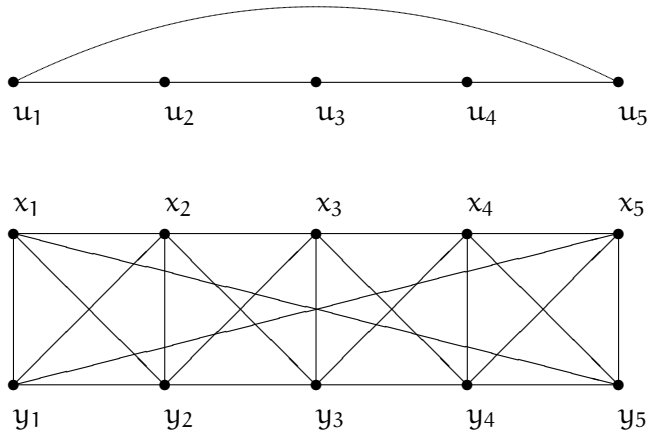

Figure 1: The construction in the proof of Theorem 4 when $k = 3$.

We claim that $\omega(L^{(k)}) \leq 4$.

In order to prove this claim let us assume the contrary that $\omega(L^{(k)}) \geq 5$. Let Δ be a 5-clique in $L^{(k)}$. Note that a set $\{x_i, y_i\}$ may contain at most 2 nodes of the clique Δ. It follows that there must be at least 3 distinct values of i such that the set $\{x_i, y_i\}$ contains at least one node of the clique Δ. For the sake of definiteness let us suppose that each of the sets

$$\{x_\alpha, y_\alpha\}, \quad \{x_\beta, y_\beta\}, \quad \{x_\gamma, y_\gamma\}$$

contains a node of the clique Δ and z_α, z_β, z_γ are the nodes of the clique Δ for which

$$z_\alpha \in \{x_\alpha, y_\alpha\}, \quad z_\beta \in \{x_\beta, y_\beta\}, \quad z_\gamma \in \{x_\gamma, y_\gamma\}.$$

Here α, β, γ are pair-wise distinct elements of the set $\{1, \ldots, n\}$.

The unordered pair $\{z_\alpha, z_\beta\}$ can be an edge of the graph $L^{(k)}$ only if the unordered pair $\{u_\alpha, u_\beta\}$ is an edge of the graph $M^{(k)}$. From this observation it follows that the nodes u_α, u_β, u_γ are the nodes of a 3-clique in $M^{(k)}$. But the Mycielski graph $M^{(k)}$ does not contain any 3-clique since $\omega(M^{(k)}) \leq 2$.

This contradiction completes the proof of the claim.

Next we claim that $\chi_{(e)}(L^{(k)}) \geq k$.

In order to prove the claim let us assume on the contrary that $\chi_{(e)}(L^{(k)}) \leq k-1$. Let E be the set of edges of $L^{(k)}$. Further let $f : E \to \{1, \ldots, k-1\}$ be the

n	$\|V\|$	$\|E\|$	node	edge	estimate
3	27	189	6	10	5
4	64	1296	12	37	9
5	125	5500	20	113	15
6	216	17550	30	273	23
7	343	46305	42	565	34
8	512	106624	56	1063	46
9	729	221616	72	1807	60
10	1000	425250	90	2922	76
11	1331	765325	110	4477	95
12	1728	1306800	132	6602	115
13	2197	2135484	156	9390	137
14	2744	3362086	182	12998	161
15	3375	5126625	210	17600	188

Table 1: Monotonic matrices. The 2nd and 3rd columns contain the number of nodes and edges of the graphs. The estimates of the clique size are in the 4th and 6th columns.

map which defines a legal coloring of the edges of $L^{(k)}$ using at most $k-1$ colors. Guided by the map f we construct a coloring of the nodes of the graph $M^{(k)}$. Let us set $U = \{u_1, \dots, u_n\}$ and let us consider the map $h : U \to \{1, \dots, k-1\}$ defined by $h(u_i) = f(\{x_i, y_i\})$ for each i, $1 \le i \le n$.

At this point we should observe that the map h defines a legal coloring of the nodes of the graph $M^{(k)}$. The only thing which needs verification is that if u_i and u_j are distinct adjacent nodes of the graph $M^{(k)}$, then the inequality $h(u_i) \neq h(u_j)$ must hold.

Since u_i and u_j are adjacent nodes in the graph $M^{(k)}$, the nodes x_i, y_i, x_j, y_j are the nodes of a 4-clique in the graph $L^{(k)}$. As the map f defines a legal coloring of the edges of the graph $L^{(k)}$, it follows that $f(\{x_i, y_i\}) \neq f(\{x_j, y_j\})$. Using

$$h(u_i) = f(\{x_i, y_i\}) \quad \text{and} \quad h(u_j) = f(\{x_j, y_j\})$$

we get $h(u_i) \neq h(u_j)$, as required.

Therefore the map h describes a legal coloring of the nodes of the graph $M^{(k)}$. In this coloring at most $k-1$ colors occur. But this is not possible since $\chi(M^{(k)}) \ge k$.

This contradiction completes the proof of the claim. □

n	$\|V\|$	$\|E\|$	node	edge	estimate
3	8	9	2	1	2
4	16	57	4	6	4
5	32	305	8	17	6
6	64	1473	14	60	11
7	128	6657	26	221	21
8	256	28801	50	875	42
9	512	121089	101	3406	83
10	1024	499713	199	13081	162
11	2048	2037761	395	49268	314
12	4096	8247297	782	186246	610

Table 2: Deletion error detecting codes. The 2nd and 3rd columns contain the number of nodes and edges of the graphs. The estimates of the clique size are in the 4th and 6th columns.

n	$\|V\|$	$\|E\|$	node	edge	estimate
6	15	45	4	3	3
7	35	385	10	23	7
8	70	1855	20	107	15
9	126	6615	35	391	28
10	210	19425	56	1131	48
11	330	49665	84	2754	74
12	495	114345	120	5918	109
13	715	242385	165	11610	152
14	1001	480480	220	21172	206
15	1365	900900	286	36514	270
16	1820	1611610	364	60054	347
17	2380	2769130	455	95038	436
18	3060	4594590	560	145441	539

Table 3: Johnson codes. The 2nd and 3rd columns contain the number of nodes and edges of the graphs. The estimates of the clique size are in the 4th and 6th columns.

4 A complexity result

In this section we will be interested in the computational complexity of the following problem.

Problem 5 *Given a finite simple graph* G *and given a positive integer* k. *Decide if the edges of* G *admit a legal coloring using* k *colors.*

We will show that for $k \geq 6$ Problem 5 is NP hard. The intuitive meaning of this result is that finding the optimal number of colors in the edge coloring of a given graph is computationally hard. Thus in practical computations we should approximate the edge chromatic number of a graph instead of exactly computing it.

We will show that Problem 5 can be reduced to the following problem which is known to be NP-complete. (See [7].)

Problem 6 *Given a finite simple graph* G *and given a positive integer* k. *Decide if the nodes of* G *have a legal coloring using* k *colors.*

The reduction of Problem 5 to Problem 5 can be accomplished using an algorithm which runs in polynomial time and uses polynomial size memory.

Let $M^{(k)}$ be the Mycielski graph with parameter k and let $e = \{x, y\}$ be an edge of $M^{(k)}$. We delete the edge e from $M^{(k)}$ but we do not delete any of the end points of the edge e. We denote the resulting graph by $N^{(k)}$.

Lemma 7 *For the graph* $N^{(k)}$ *defined above the following holds.*

(1) The nodes of the graph $N^{(k)}$ *can be colored legally using* $k-1$ *colors.*

(2) In each legal coloring of the nodes of the graph $N^{(k)}$ *using* $k-1$ *colors the nodes* x *and* y *must receive the same color.*

Proof. The Mycielski graph $M^{(3)}$ is a circle consisting of 5 nodes and 5 edges. After deleting an edge from $M^{(3)}$ the nodes of the remaining graph can be colored legally with 2 colors. The end points of the deleted edge must receive the same colors since otherwise one puts back the deleted edge and the nodes of the graph $M^{(3)}$ could be legally colored with 2 colors. Thus the special case $k = 3$ is settled.

The Mycielski graph $M^{(4)}$ has 11 nodes $v_1, \ldots, v_5, u_1, \ldots, u_5, w$. The set of nodes $\{v_1, \ldots, v_5\}$ induces a subgraph H in $M^{(4)}$ such that H is isomorphic to $M^{(3)}$. Figures 2 and 3 illustrate this step of the proof.

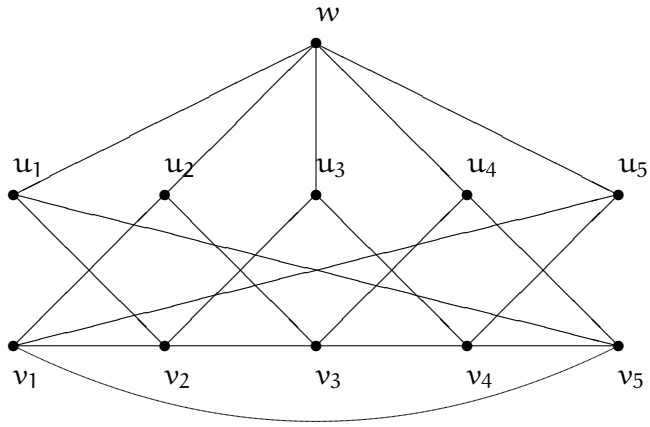

Figure 2: The Mycielski graph $M^{(4)}$.

The node u_i is adjacent to the neighbors of v_i for each i, $1 \leq i \leq 5$. The node w is adjacent to u_i for each i, $1 \leq i \leq 5$. After deleting the edge $\{v_1, v_5\}$ from H the nodes of the resulting graph H' can be legally colored with 3 colors. To exhibit a legal coloring of the nodes let us color the nodes $u_1, \ldots, u_5$ with colors 1. The nodes $v_1, \ldots, v_5$ can be colored with the colors 2, 3. Finally, we color the node w with color 3. We have ended up with a legal coloring of the nodes of the graph $N^{(4)}$ using 3 colors. This settles the case $k = 4$.

The Mycielski graph $M^{(5)}$ has 23 nodes $v_1, \ldots, v_{11}, u_1, \ldots, u_{11}, w$. The set of nodes $\{v_1, \ldots, v_{11}\}$ induces a subgraph H in $M^{(5)}$ such that H is isomorphic to $M^{(4)}$. The node u_i is connected by an edge to each of the neighbors of the node v_i for each i, $1 \leq i \leq 11$. Finally, we connect the node w to the node u_i by an edge for each i, $1 \leq i \leq 11$. The graph H has an edge e such that after deleting e from H the nodes of the resulting graph H' can be colored legally with 3 colors.

We assign color 1 to node u_i for each i, $1 \leq i \leq 11$. The nodes $v_1, \ldots, v_{11}$ can be colored legally with 3 colors. We will use the colors 2, 3, 4. Finally, we assign color 4 to node w. This provides a legal coloring of the nodes of the graph $N^{(5)}$ using 4 colors. Therefore the case $k = 5$ has been settled.

After working out the $k = 5$ special case we are well prepared to settle the general case using an induction on k.

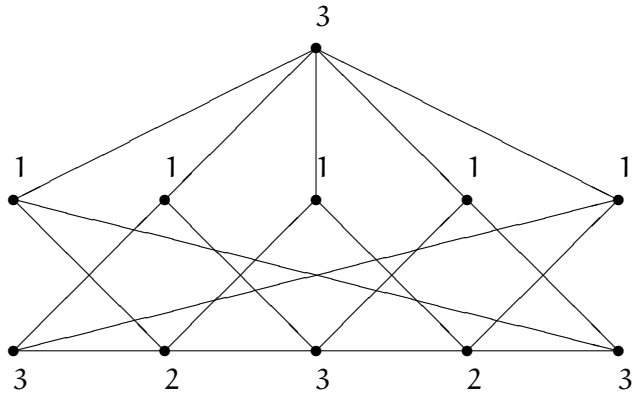

Figure 3: The graph $N^{(4)}$ with colored nodes.

Set $V = \{v_1, \ldots, v_n\}$ and $U = \{u_1, \ldots, u_n\}$. Let $U \cup V \cup \{w\}$ be the set of nodes of the Mycielski graph $M^{(k)}$. The set of nodes V induces the subgraph H which is isomorphic to $M^{(k-1)}$. Note that $N(w) = U$ and $N(u_i) \cap V = N(v_i) \cap V$ for each i, $1 \leq i \leq n$.

By the inductive assumption H has an edge e such that after deleting e from H the nodes of the resulting graph H' can be colored legally using $k-2$ colors.

We color each node in U with color 1. The nodes in V can be colored legally using the colors $2, \ldots, k-1$. The node w can be colored with color $k-1$. In this way we ended up with a legal coloring of the nodes of N^k using $k-1$ colors.

Putting back the edge e gives back the graph $M^{(k)}$. The nodes of $M^{(k)}$ can be colored legally with k colors but not with $k-1$ colors. Consequently, the end points of the edge e must receive the same colors in any legal coloring of the nodes of N^k using $k-1$ colors.

This completes the proof. □

Using the graph $N^{(k)}$ we construct a new graph $L^{(k)}$ by adding a new edge $\{z, x\}$ to $N^{(k)}$. Figure 4 depicts the graph $L^{(k)}$. The newly constructed graph $L^{(k)}$ clearly has the following properties.

(1) The nodes of the graph $L^{(k)}$ can be colored legally using $k-1$ colors.
(2) In each legal coloring of the nodes of the graph $L^{(k)}$ using $k-1$ colors the nodes z and y must receive distinct colors.

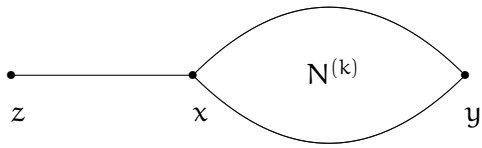

Figure 4: The graph $L^{(k)}$.

(3) If the nodes z and y are colored with distinct colors, then this partial coloring of the nodes of $L^{(k)}$ can be extended to a legal coloring of the nodes of $L^{(k)}$.

In order avoid notational difficulties first we will deal with the k coloring problem in the special case when $k = 6$ and so we will use the graph $L^{(7)}$ as an auxiliary graph. The reader can verify that the graph $L^{(7)}$ has 95 nodes and 640 edges.

Let $G = (V, E)$ be a finite simple graph and let $u_1, \ldots, u_n$ be the edges of G. From the given graph G we construct a new graph G'.

Let $L_{i,j}$ be an isomorphic copy of the graph $L^{(7)}$. We assume that $w_{i,j,1}, \ldots, w_{i,j,95}$ are all the nodes of the graph $L_{i,j}$ and the node $w_{i,j,1}$ of $L_{i,j}$ corresponds to the node z of the graph $L^{(7)}$. Further we assume that the node $w_{i,j,95}$ of $L_{i,j}$ corresponds to the node y of the graph $L^{(7)}$.

From the given graph $G = (U, E)$ we construct a new graph G'. Let the unordered pair $\{u_i, u_j\}$ be an edge of the graph G. We replace this edge of G by $L_{i,j}$. We identify the node u_i of G with the node $w_{i,j,1}$ of the graph $L_{i,j}$ and we identify the node u_j of G with the node $w_{i,j,95}$ of the graph $L_{i,j}$. In the graph $L^{(k)}$ the roles of the nodes y and z are not symmetric. In order to avoid ambiguity in the construction we assume that for the edge $\{u_i, u_j\}$ the condition $i < j$ holds.

From the graph G' we construct a new graph Γ. Let us suppose that $w_1, \ldots, w_m$ are all the nodes of G'. The graph G' has m nodes and the graph Γ will have $2m$ nodes $x_1, \ldots, x_m, y_1, \ldots, y_m$. To the node w_i of G' we assign an edge $\{x_i, y_i\}$ of the graph Γ for each i, $1 \leq i \leq m$. Next if the unordered pair $\{w_i, w_j\}$ is an edge of G', then we add the edges

$$\{x_i, x_j\}, \quad \{x_i, y_j\}, \quad \{y_i, x_j\}, \quad \{y_i, y_j\}$$

to Γ.

The pivotal result of our consideration in pursuing the $k = 6$ particular case is summarized in the following assertions.

Theorem 8 *For the graph Γ defined above the following holds.*

(1) If the edges of the graph Γ have a legal coloring using 6 colors, then the nodes of the graph G have a legal coloring using 6 colors.

(2) If the nodes of the graph G admit a legal coloring with 6 colors, then the edges of the graph Γ admit a legal coloring with 6 colors.

Proof. In order to prove claim (1) let us assume that the edges of the graph $\Gamma = (W, F)$ have a legal coloring using 6 colors. Let us suppose that the map $f : F \to \{1, \ldots, 6\}$ describes this coloring. Using this coloring of the edges of the graph Γ we construct a coloring of the nodes of the graph G. To the node u_i of G we assign the colors of the edge $\{x_i, y_i\}$ of Γ. In other words we define a map $g : U \to \{1, \ldots, 6\}$ by setting $g(u_i)$ to be equal to $f(\{x_i, y_i\})$.

The map g describes a coloring of the nodes of the graph G using 6 colors. We claim that g describes a legal coloring of the nodes of the graph G. In order to verify the claim it is sufficient to show that if the unordered pair $\{u_i, u_j\}$ is an edge of the graph G, then $g(u_i) \neq g(u_j)$ must hold. The node u_i of G corresponds to the node $w_{i,j,1}$ of the graph G' and the node u_j of G corresponds to the node $w_{i,j,95}$ of the graph G'. The node $w_{i,j,1}$ corresponds to the edge $\{x_{i,j,1}, y_{i,j,1}\}$ of the graph Γ. Similarly, the node $w_{i,j,95}$ corresponds to the edge $\{x_{i,j,95}, y_{i,j,95}\}$ of the graph Γ.

We know that the edges $\{x_{i,j,1}, y_{i,j,1}\}$ and $\{x_{i,j,95}, y_{i,j,95}\}$ receive distinct colors, that is, $f(\{x_{i,j,1}, y_{i,j,1}\}) \neq f(\{x_{i,j,95}, y_{i,j,95}\})$. Consequently $g(u_i) \neq g(u_j)$, as required.

In order to prove assertion (2) let us assume that the nodes of G have a legal coloring with 6 colors. We assume that the map $g : U \to \{1, \ldots, 6\}$ describes this coloring. Suppose that the unordered pair $\{u_i, u_j\}$ is an edge of the graph G and $i < j$ holds.

When we constructed the graph G' from the graph G we replaced the edge $\{u_i, u_j\}$ of G by $L_{i,j}$ which is an isomorphic copy of the graph $L^{(7)}$. Let $x_{i,j}$, $y_{i,j}$, $z_{i,j}$ be the nodes of the graph $L_{i,j}$ that correspond to the nodes x, y, z of the graph $L^{(7)}$ at the isomorphism.

The node $z_{i,j}$ of the graph G' gets the color of the edge u_i of the graph G. The node $y_{i,j}$ of the graph G' gets the color of the edge u_j of the graph G. The node $x_{i,j}$ of the graph G' gets the color of the edge x of the graph G. We know that this partial coloring of the nodes of the graph $L_{i,j}$ can be extended to the coloring of all nodes of the graph $L_{i,j}$.

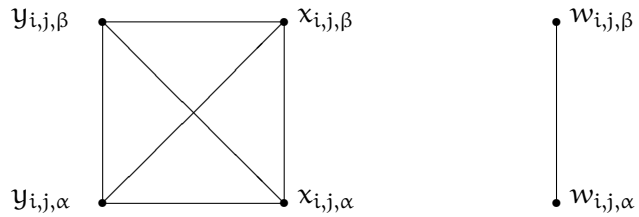

Figure 5: The 4-clique Δ in the graph $L_{i,j}$.

For the sake of a convenient notation we rename the nodes of the graph $L_{i,j}$. Let $w_{i,j,1}, \ldots, w_{i,j,95}$ be the nodes of the graph $L_{i,j}$. We will assume that

$$w_{i,j,1} = z_{i,j}, \quad w_{i,j,2} = x_{i,j}, \quad w_{i,j,95} = y_{i,j}.$$

When we constructed the graph Γ from the graph G' we have assigned an edge $\{x_{i,j,\alpha}, y_{i,j,\alpha}\}$ of Γ to the node $w_{i,j,\alpha}$ of G' for each i, j, α, $1 \leq i < j \leq m$, $0 \leq \alpha \leq 95$. Now we assign the color of the node $w_{i,j,\alpha}$ of the graph G' to the edge $\{x_{i,j,\alpha}, y_{i,j,\alpha}\}$ of the graph Γ.

When the nodes $w_{i,j,\alpha}$ and $w_{i,j,\beta}$ were adjacent in the graph G', then we added the edges

$$\{x_{i,j,\alpha}, x_{i,j,\beta}\}, \quad \{x_{i,j,\alpha}, y_{i,j,\beta}\}, \quad \{y_{i,j,\alpha}, x_{i,j,\beta}\}, \quad \{y_{i,j,\alpha}, y_{i,j,\beta}\} \tag{2}$$

to the graph Γ during the construction of the graph Γ from the graph G'. The situation is shown by Figure 5. The edge $\{x_{i,j,\alpha}, y_{i,j,\alpha}\}$ of the graph Γ receives the color of the node $w_{i,j,\alpha}$ of the graph G' and the edge $\{x_{i,j,\beta}, y_{i,j,\beta}\}$ of the graph Γ receives the color of the node $w_{i,j,\beta}$ of the graph G'.

The nodes of G' are colored using 6 colors. We intend to colors the edges of Γ using these 6 colors. Out of the 6 colors we intend to use for the coloring the edges of the graph Γ 4 colors are still available to color the edges (2). In this way we get a legal coloring of the edges of the graph Γ. □

The main result of this section is the following theorem.

Theorem 9 *Problem 5 is NP hard for each integer* $k \geq 6$.

Proof. Theorem 8 settles the special case $k = 6$. For the remaining part of the proof we assume that $k \geq 7$. We start with the graph $N^{(k+1)}$ and follow a reasoning analogous to the proof of Theorem 8.

A few elementary estimates are still missing to complete the argument. Let $G = (V, E)$, $G' = (V', E')$, $\Gamma = (W, F)$. It is clear that there is a positive constant c_1 such that $|V'| \leq c_1|V|$. One can choose c_1 to be the number of nodes of the auxiliary graph $L^{(k+1)}$. There is a positive constant c_2 for which $|E'| \leq c_2|V|^2$. Indeed,

$$|E'| \leq (1/2)|V'|^2 \leq (1/2)c_1^2|V|^2 = c_2|V|^2.$$

The computation

$$|W| \leq 2|V'| = 2c_1|V| = c_3|V|$$

shows that there is a positive constant c_3 such that $|W| \leq c_3|V|$. Finally, there is a positive constant c_4 with the property that $|F| \leq c_4|V|^2$. This can be seen from

$$|F| \leq 6|E'| \leq 6c_2|V|^2 = c_4|V|^2.$$

The essential point is that the quantities $|E|$, $|W|$, $|F|$ can be over estimated by a polynomial in terms of $|V|$. As we can see the degree of this polynomial is two. The leading coefficient can be very large. As a matter of fact it is an exponential function of k. But for each fixed k the leading coefficient is a constant. □

Acknowledgements

We would like to express our thanks for the anonymous referee.

References

[1] I. M. Bomze, M. Budinich, P. M. Pardalos, M. Pelillo, The Maximum Clique Problem, Handbook of Combinatorial Optimization Vol. 4, Kluwer Academic Publisher, 1999. ⇒64

[2] R. Carraghan, P. M. Pardalos, An exact algorithm for the maximum clique problem, *Operation Research Letters* **9** (1990) 375–382. ⇒67

[3] J. Hasselberg, P. M. Pardalos, and G. Vairaktarakis, Test case generators and computational results for the maximum clique problem, *Journal of Global Optimization* **3** (1993) 463–482. ⇒64

[4] D. Kumlander, *Some Practical Algorithms to Solve the Maximal Clique problem* PhD. Thesis, Tallin University of Technology, 2005. ⇒64

[5] C. Morgan, *A Combinatorial Search with Dancing Links*, PhD. Thesis, Univ. of Warwick, 1999–2000. ⇒64

[6] J. Mycielski, Sur le coloriage des graphes, *Colloq. Math.* **3** (1955) 161–162. ⇒70

[7] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley Publishing Company, Inc., 1994. ⇒64, 74

[8] S. Szabó, Parallel algorithms for finding cliques in a graph, *Journal of Physics: Conference Series* **268** (2011) 012030 ⇒63, 65

Received: February 16, 2016 • Revised: April 17, 2016

DE GRUYTER OPEN
Acta Univ. Sapientiae, Informatica **8,** 1 (2016) 82–95

DOI: 10.1515/ausi-2016-0005

Improving the effectiveness of FMEA analysis in automotive – a case study

Gábor VÁNYI
Eötvös Loránd University, Budapest
email: vanyig@ceasar.elte.hu

Abstract. Many industries, for example automotive, have well defined product development process definitions and risk evaluation methods. The FMEA (Failure Mode and Effects Analysis) is a first line risk analysis method in design, which has been implemented in development and production since decades. Although the first applications were focusing on mechanical and electrical design and functionalities, today, software components are implemented in many modern vehicle systems. However, standards or industry specific associations do not specify any "best practice" how to design the *interactions* of multiple entities in one model. This case study focuses on modelling interconnections and on the improvement of the FMEA modelling process in the automotive. Selecting and grouping software components for the analysis is discussed, but software architect design patterns are excluded from the study.

1 Introduction

Today, software working all over our vehicles, from sensors and cameras to navigation to infotainment systems to diagnostics. In 2001, cars had a minimal amount of code in them, nowadays, a new car has about 100 million lines of code. The increase of software parts reduces weight, involves cost optimization and decreases delivery time compared to similar complex mechanic

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 46S99
Key words and phrases: FMEA, risk analysis, system modelling, automotive

or hardware changes in the development life-cycle. Complexity of hardware, software and mechanical systems exponentially increase with the possibility of harm and unwanted side effects. Hence, different standards implemented for reliability analysis and risk estimation.

The FMEA method was introduced in the automotive industry based on the results of the Apollo airspace program in 1970s [6]. It was the first time when this method could show opportunity to find possible failures in big and complex systems like a space shuttle. This method is widely used both in development and manufacturing. For the automotive industry the QS 9000 standard and the SAE J1739 defines the implementation. Effective analysis of software components remain a challenge, because it cannot be assessed like hardware. During the analysis a possible question can be: "what kind of harm can the development environment, programming language, compiler, coding method (pointer, timer or operator) cause?". If the functionality of software can be analysed by itself, it also generates questions like "what part of the code is a function?" and "what are the inputs and outputs of the given block?". And finally, "what kind of failures and causes can be considered during the analysis?". If a well-defined structure was used, then the effects of a failure can be traced back through the whole system. This streamlined approach can motivate technical risk analysis to handle each unwanted risk in full detail, hence project managers will see the quality and value together in their product.

2 FMEA overview

The following FMEA types are used in automotive: (1) system, (2) design and (3) process. These can be used in different logical levels. For example, system FMEA should be used for a subsystem, design FMEA for a simple screw, and process FMEA to evaluate risks in manufacturing. There is hierarchical relation between system, design and process FMEA (in this order). Two possible additional levels can be used to collect failure effects from the top level and one additional level on the lowest part (as cause level) which can be used as a failure cause catalogue.

The risk ranking is based on risk priority number (RPN), which can be calculated as the multiplication of parameters Severity (S), Occurrence (O) and Detection (D). The ranking process can be formal, since evaluating catalogues facilitates teams in finding the right risk evaluation values (see i.e. SAEJ1739, VDA, etc.). Moreover, proper structuring of technical systems and traceability of design changes is more challenging than risk evaluation. Former version of

VDA (released in 2009) required that every risk have to be mitigated, where RPN is larger than a certain threshold [9]. The newest version raises attention to S and O, thus, it defines a matrix of S and O for more detailed risk ranking. Many FMEA expert states that these matrices may not show the real practical risk. Multiplication of a high S value with a low O values means that very critical failure may not happen very often, but when it happens, it can cause high damage. Thus, some companies decided that they try to reduce their potential risks to as low levels as possible with different actions by tests, design changes or reviews together with their customers during development. A very important rule of thumb is to evaluate severity values on a highest level and inherit these to lower levels. This can ensure that a failure will have the same meaning of seriousness in the whole system and will be encountered and managed.

Beyond the effectiveness, sufficient level of technical content has significant differences from the quality point of view, because these depend on human factors. The electric, mechanic and electro-pneumatic systems have to be analysed by different methods of reliability and risk analysis in parallel. If these were coordinated as an unified systematic method, it would reduce capacity and time requirements. The FMEDA (Failure Mode and Effects and Diagnostic Analysis [8]) principles and purposes can be used in general FMEA as well. Many articles exist about software component analysis (see e.g. [4]), but these do not tend to explain how to connect software components with electrical hardware and mechanical interfaces.

3 Creating FMEAs

The “5 step method” is a well-known practice in the automotive industry. It has been introduced in the VDA standard [10][5]. This method prefers to start by creating hierarchical groups of system element networks, then connect functions to each system element, define the effects of failure operations, evaluate risks, and finally rank and mitigate risks. Although this method is well-known, the resource capacity is very high because of the high number of reviews for the newly developed product. Therefore ”best practices” and internal know-how have been used at many companies to quick start the process as a kind of template, but even in this case many redundant steps cannot be easily eliminated. Companies with many independent departments (or competent centres) usually use different strategies for system modelling and focuses different points of analysis. These methods generate latent quality gaps and

risks. Another problem comes, when different products or components connecting each other into one large system, thus FMEAs shall be connected as well. Fortunately, it is feasible to have a common consensus regarding which processes will be used on quality side. On engineering side, the modelling is supported by P-diagrams and boundary diagrams, which are introduced in QS 9000 standard [7]. Certain failures can be detected easier both by Fault Tree Analysis (FTA) or Functional Hazard Analysis (FHA) methods for the first step.

Hardware and mechanic elements can be analysed easier than software elements, because they can be bounded physically and standards support many methods for analysing. Software parts modelled mainly in Unified Modelling Language (UML) or flowcharts. Many tools support these forms but identifying safety gaps or risks are not so easy in safety critical or safety related modules and functions in the same way as estimating their effects in case of harms. Interfaces, built-in parameters and data layer functions can cause many exceptions if they have wrong values (programmed) or have been intentionally modified. Customers usually want to see identified risks under consideration, but it can be difficult to analyse and present thousands of combinations of values. An optimization strategy can be when software module analyses includes functional or logical grouped evaluation of interfaces. These cases are hard to find by tests of course, but FMEA should support identifying relevant requirements and functions which must be examined later.

A complex automotive system should be divided into hardware, software and mechanic components, since they cannot be analysed in the same time due to the different scheduling of development. The highest level is common, it is usually used for the effect level or system FMEA level. Using this common effect level has an advantage, since each severity number can be discussed with the customer, plus these failures can be guided easily through the system from top (requirement) down to an element (i.e. screw or software module). Later on, if an effect line of an identified risk was known, the failure network and function network would show these effect lines. If separated effect level was used only, it would cause quality gaps and shortcoming of risk evaluations on lower levels in the system.

System FMEA is the first point in the analysis where full risk evaluation is performed. There can be several system FMEA applied on the same level in parallel. This level lists the functions of the system. They are connected to one level higher to receive severity ranking for each failure as failure effect from the effect level.

One level below the design FMEAs can be found, where mechanical, elec-

trical elements and software components analysed in logical connection to the higher levels system FMEAs. This makes a logical network to overview which components participate in the given function. For example, the system level analysis may show an increasing pressure in chamber, the design level may connect a piston to a housing, etc., which are part of this function in the compressing air. If a system specialist was able to answer the question "what happens if this rod causes a failure?" then a failure network would help to see the points of possible failures and related functions up to the top-effect level. The last level is the process FMEA, which is used to evaluate the risk of components production.

Finally, the lowermost part is the cause level. It is not regularly used, but has many advantage if design or process failure causes are handled in a common failure catalogue.

3.1 Effect level

As it was discussed before, this is the top level in the FMEA, even if the three level rule is applied. Groups of the agreed first level requirements are listed in the function column. These are usually declarations of dimensions, some important internal requirements including internal lessons learned knowledge, specifications of standards, results of safety analysis (i.e. FHAs Top Effects) and regulations from relevant important laws. Possible failures and harms of requirement violation are evaluated in the next column. These are evaluated by system experts, safety professionals and the moderator. Usually ranked and evaluated the severity numbers together or at least reviewed with the customer who shall approve them. These targets of safety goals and functions shall be reached during the development. Safety is a killer criteria which shows how the unwanted actions are handled, since severity values cannot be lower than 9, but usually 10. Lower level (system) FMEA is connected to causes, these show the affected functions of each failure.

Functions	Potential failure	Effect	S	Cause	O	P/D action	D
Braking	Speed is not reduced		10	ABS system			
Steering	Unintended maneuver		10	EPS system			
Comfort	Air conditioning fails		5	Cooling system			

Table 1: Effect level example

3.2 System FMEA levels

This is the first level where full risk analysis has to be performed. The risk assessment of system level shows weak points of system functionalities and helps to eliminate these gaps. This level usually has only one FMEA in case of less complicated systems with only a small number of functionalities and components. If the system has more advanced electronic, mechanical and software parts then various types of system FMEA have to be performed parallel in different content. This method will support sorting elements for a better system overview and handling scheduled delays according to different development life-cycles.

System level FMEA approach consists of two different contents. One of them focuses on components instead of functionalities, because production uses these information. Thus, the characteristics of strict suiting dimensions and material definitions have more value in labelling special characteristics (S/C or C/C) since these parts have more strict regulations for quality and product security. This approach should be applied for mechanical/pneumatic parts, in which case system level FMEA lists groups of mechanical functions (i.e. linkage group – containing functionalities for coupling two parts). One level below, the design FMEAs will list components which have different roles in this functionality. A component can be connected to more system FMEA if it is affected in that functionality.

Very significant difference from other mechanical FMEAs is declaring special characteristics on system levels. Because change of material, component or dimension will be traced easier on system level rather than design level, especially when design FMEA will be obsoleted or used in another production.

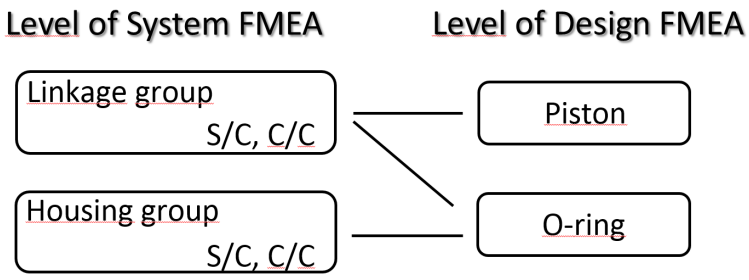

Figure 1: System FMEA connection to design FMEAs – mechanical and pneumatic components

The other approach is more beneficial for electronic and software components. This structure contains one system FMEA for electronic hardware and another one for software. If software components are too complicated then they should be divided into logical groups where each group has one common system FMEA.

One level below design FMEA is listed. Each component of an electronic circuit should be listed and connected to system level, showing which one participates in a given function. Usually short-cut, opening or missing component shall be examined. If these cases of failures examined, then they would have good input for FMEDA as well.

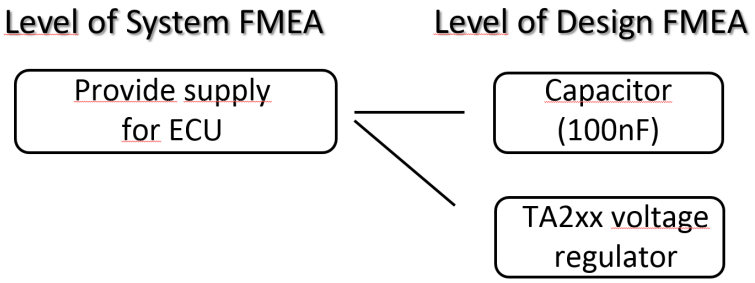

Figure 2: System FMEA connection to design FMEAs – electronic hardware

Software elements shall be analysed similarly, but the author experienced that three layers of software modelling will have more advantages. These free layers are (1) system level functions (high level functionality), (2) data transmission layer (communication arrays between modules) and (3) platform (low level functions, directly connected to hardware level). These are on the same level (design), but connected to each other via their interfaces.

Software modules are able to connect one element to more, thus the rule of the mechanic parts or hardware parts can be applied here. However, special characteristics (S/C, C/C) are not applicable because hardware components are examined for many times during the production (i.e. End–of–Line–Testing, testing each component at manufacturer, etc.), software parts are tested during development life-cycle and will have similar functionalities as have before in the production. Thus, final assembly test must validate that the assembled product is valid and working according to the specifications.

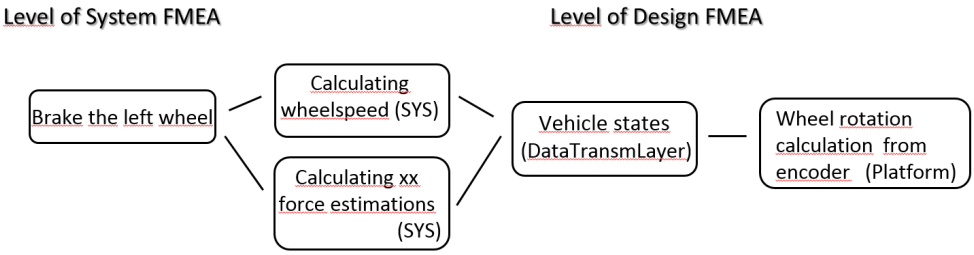

Figure 3: System FMEA connection to design FMEAs – software components

3.3 Design FMEA levels

This level is used for analysing disciplinary designs of hardware, mechanic, pneumatic or software components. Evaluation is made by technical experts, test engineers and an FMEA moderator. The notion "function" defines different meaning in each disciplines, but the top effect level can be similar. Full risk evaluation can be made on this levels as well.

In case of mechanical discipline for every element in the BOM (Bill of Material) a separate design FMEA have to be created. These elements related to the manufacturing process, thus data shall be assessable for production FMEA. Special characteristics such as S/C (Significant Characteristic) or C/C (Critical Characteristic) are identified for handling important dimensions or material definitions during manufacturing. The question is how special characteristics can be defined correctly, because there are no feedbacks from production of the given designs. A pre-defined a template with some technical points support this evaluation (see Table 2). The project team evaluates the template forms severity and occurrence values, then special characteristics are defined for the system level function, as it introduced earlier. This action makes easier to transfer the given function into the new product and supports to re-use them.

Hardware analysis is possible via the following the signal path from a single pin to the controllers software. It requires additional resources from software beside hardware developers. Analysis of electronic modules will have more advantage if FMEDCA (Failure Modes, Effects and Diagnostic Coverage Analysis) can be covered. PCB (Printed Circuit Board) contains many elements, hence these will make many extra work during the FMEA design process. Possible solution would be grouping them according to their functions, like power supply, etc. Short-cuts and cut in the circuit are usually analysed, failure cause

Function	Potential Failure
Ensure appropriate material properties	
	Wrong dilatation coefficient defined
	Wrong tribological properties defined
	Wrong E-modulis defined
	Wrong hardness defined
	Wrong braking strain defined
	Wrong Shore hardness defined
	Wrong glass transition temperature defined
	Wrong Poisson ratio defined
	Wrong Shear modulus defined
	Wrong property class defined
	Wrong compression set defined
	Wrong basic production technology defined
	Wrong Yield stress defined
Ensure appropriate geometrical properties	
	Wrong alignment relative to connecting components defined
	Wrong length relative to connecting components defined
	Wrong width relative to connecting components defined
	Wrong depth relative to connecting components defined
	Wrong thickness relative to connecting components defined
	Wrong diameter relative to connecting components defined
	Wrong greased area defined
	Wrong spring parameters defined
Ensure appropriate surface properties	
	Wrong surface coating defined
	Wrong roughness depth defined
	Wrong accuracy class for roughness
	Wrong surface treatment defined
	Wrong rill direction defined
	Wrong percentage contact area defined
Ensure appropriate connection of components	
	Wrong fastening element type selected
	Wrong numbers of fastening element defined
	Wrong fastening element distribution defined
	Wrong fastening torque defined
	Wrong fastening force defined
	Wrong fastening order defined
	Wrong connecting geometry defined

Table 2: Form sheet for mechanic design FMEA to support identification of special characteristics

catalogue on one level below supports finding design or other kind of possible failures.

Modelling the functionality of software modules is sometimes a kind of adventure. Teams usually facing with difficulties while performing analysis on pure software components, such as monitoring or continuously running routines. One of the lessons learned of FMEA moderation should be avoiding to create pure software FMEAs, because failure causes usually lead to the limits of programming language, edges of development environment or coding errors. Better if the analysis focuses on functionality of software modules. Requirements of the development level (RQ3 domain) are good starting points but calculations, actuations or any larger functionality should be considered. Causes of failures usually connected to other software modules, preventive actions usually refers violation of development processes or unintended failure of monitoring routines.

3.4 Cause levels

This is the bottom-most level of the FMEAs. The forms are not evaluated as design or system FMEAs, but they has been used for a kind of failure catalogue. Although, it also happens that this level is not used, but FMEA holds every necessary information and evaluation. We used this level to collect common disciplinary failures, like causes of design failures for evaluation support.

4 Increasing review efficiency

Motivating people to actively participate on FMEA review meetings is a big challenge. Finding the right information for FMEA and deciding if it is really the right one is also hard. The capacity usually limited and projects usually facing with time pressure, thus, FMEA meeting shall be optimized and speeded up to an efficient level. Minimal capacity of a meeting requires at least two experts of the given component for ensuring the right technical understanding and review each other. An FMEA moderator is usually necessary inviting test engineers for field experiences. Then, a thematic questionnaire and form sheets can support optimizing time and efficiency.

Typical questions are collected for thought-provoking here in order to support (i.e. software) FMEAs. General experience shows that developers are thinking usually in their modules inside, but has no idea what will they receive on input. Introducing or thinking over functionalities and newly devel-

oped or modified modules is a first step for placing this module in the system hierarchy. Then, the input and output of the given components can be examined, listening on keywords like "default value" and "NA" state. Afterwards questions can be stated about safe state if it is relevant or about scheduling operation. Thinking about these data the following questions will support you finding your own questions. Some examples were enumerated:

1. Assess the interfaces:
    - Which modules are using the output of this module and what are the input?
    - What are the default values, what happens if 0 or N/A occurs?
    - Is there any declaration for combination of value pairs on input or output?
    - Are there any configuration parameters which have been used for calculation?
2. Check the functionality of the module:
    - What kind of calculations have been made?
    - Is it possible that unintended overflow or underflow causing safety critical event?
    - Is it assured that unexpected values from other modules have been handled?
    - What thresholds have been examined or data comparison have been used?
    - What have been calculated and which functionality belongs to this calculation?
3. Safety of the module:
    - Are there any plausibility checks for calculated values?
    - Are there any check or monitoring for communication lines?
    - What kind of test or diagnostic function were implemented or used?
    - What is the scheduling period?

These questions have another benefits. If somebody has been interviewed about the development and requirement this person has to think what have been done and why? The final result of this thinking is also booked in the FMEA and assured that the right person has done the right development (other experts support this walk-through as well).

5 Conclusion

Risk evaluation of the designed functions has an increasing importance in automotive. Appropriate structuring brings better understanding of system functionality, and risk evaluation gives more precise feedback for developers. Re-using of earlier developed components can be supported by grouping the appropriate elements. The very first feedback shows that the production could connect their process FMEAs to design FMEAs easier due to the fact that functional grouping helps in understanding the component identification. Surveying other solutions, many articles just focus on the right risk evaluation [2] [3] using e.g. fuzzy logic in order to support risk ranking and the evaluation mechanism.

The presented system modelling method, particularly in software structuring and mechanical design grouping, will support the daily work better. Software FMEAs usually focus on evaluation of variables and equipment analysis [1]. This case study focused on structuring software and examined the interconnections among these structured levels instead. The usage of form sheets in mechanical design FMEA may reduce capacity demand due to answering similar questions at each component part. Experiences show that the cumulative meeting time frame of software FMEAs with questionnaires have been reduced from two months to two weeks, using a twice-per-week scheduling. On the other hand, the questionnaire supported the functionality review for developers and testers focusing on communication failures between modules and result of a calculation, scheduling interactions, safe states, etc. It is part of the software review at module test level of course, but not only one module have been inspected in this case. Re-using of former FMEA contents become easier in the newer generations and variants if unified structure and content were chosen.

Form sheets in design FMEA change the mindset for mechanical designers because they have to focus more on design points instead of ranking the levels of solutions. Hence, there is no need waiting for the feedback from production failures. Evaluating the right characteristics in the beginning shall be paid attention since it has influence on the cost of the product.

Designing mechanical connections is a baseline for production since processes are connected to design's result. Then, production is able to connect their FMEAs easier while they are able to identify the sources of special characteristics, such as material, mechanical connection, edges of the components connection, etc. Later, these technical information will support investigations in product modifications.

Motivating participants for an FMEA meeting is not an easy task under the pressure of deadlines. Making the evaluation time shorter and asking the right questions in right time definitely is a dream. If quality and daily practice meet in a well organized process and modelling method set then it can realize better quality and less postpones of production start and product recalls.

The reader shall decide how will implement these ideas in his or her FMEAs. Author just demonstrated a case study without any responsibility of insufficient use of these points.

6 Acknowledgements

I should like to thank János György and Sándor Drienyovszky for their ideas and experiences together with the result formed the concept of mechanical design FMEA part. I thank the referee for providing constructive comments and help in improving the contents of this paper. Finally, I should like to thank to my supervisor, Attila Kovács for his suggestions preparing this article.

References

[1] J. H. Craig, A software reliability methodology using software sneak analysis, SW FMEA and the integrated system analysis approach , in: *Reliability and Maintainability Symposium, 2003. Annual*, IEEE, 2003, pp. 12–18. ⇒93

[2] L. Pokorádi, T. Fülep, Reliability in automotive engineering by fuzzy rule-based FMEA, in: *Proceedings of the FISITA 2012 World Automotive Congress,* Volume 197 of the series Lecture Notes in Electrical Engineering, Springer Berlin Heidelberg, 2012, pp. 793–800. ⇒93

[3] L. Pokorádi, B. Szamosi, Fuzzy Failure Modes and Effects Analysis with Summarized Center of Gravity DeFuzzification, in: *16th IEEE International Symposium on Computational Intelligence and Informatics,* CINTI 2015 , IEEE, 2015, pp. 147–150. ⇒93

[4] K. H. Pries, Failure mode & effect analysis in software development, in: *Automotive Electronics Reliability,* edited by Ronald K. Jurgen, SAE International, PT-82, 1998, pp. 351–360. ⇒84

[5] P- Urban, DFMEA acc. to VDA - 5 steps approach, *OALC Reliability Blog.* 2011, http://www.opsalacarte.com. ⇒84

[6] Society for Automotive Engineers, Design Analysis Procedure For Failure Modes, Effects and Criticality Analysis (FMECA) 1967. ARP926. ⇒83

[7] Quality System 9000 Handbook, *Volume FMEA handbook* 2006. ⇒85

[8] TüV NORD, *Failure Modes Effects and Diagnostic Analysis* (2013), http://www.tuev-nord.de/. ⇒84

[9] Verband der Automobilindustrie, Qualitätsmanagement in der Automobilindustrie, Sicherung der Qalität vor Serieneinsatz, System FMEA *VDA QMC* 4 (2006) 124–139. ⇒84
[10] Verband der Automobilindustrie, Qualitätsmanagement in der Automobilindustrie, Produkt- und Prozess-FMEA, *VDA QMC* 4 (2006) 30–63. ⇒84

Received: March 4, 2016 • Revised: April 18, 2016

DE GRUYTER OPEN Acta Univ. Sapientiae, Informatica **8,** 1 (2016) 96–107

DOI: 10.1515/ausi-2016-0006

On the nullity of a family of tripartite graphs

Rashid FAROOQ
School of Natural Sciences,
National University of Sciences and
Technology, Islamabad, Pakistan
email: farook.ra@gmail.com

Mehar Ali MALIK
School of Natural Sciences,
National University of Sciences and
Technology, Islamabad, Pakistan
email: alies.camp@gmail.com

Qudsia NAUREEN
School of Natural Sciences,
National University of Sciences and
Technology, Islamabad, Pakistan
email: naureenqudsia@gmail.com

Shariefuddin PIRZADA
University of Kashmir, Srinagar, India
email:
pirzadasd@kashmiruniversity.ac.in

Abstract. The eigenvalues of the adjacency matrix of a graph form the spectrum of the graph. The multiplicity of the eigenvalue zero in the spectrum of a graph is called nullity of the graph. Fan and Qian (2009) obtained the nullity set of n-vertex bipartite graphs and characterized the bipartite graphs with nullity $n-4$ and the regular n-vertex bipartite graphs with nullity $n-6$. In this paper, we study similar problem for a class of tripartite graphs. As observed the nullity problem in tripartite graphs does not follow as an extension to that of the nullity of bipartite graphs, this makes the study of nullity in tripartite graphs interesting. In this direction, we obtain the nullity set of a class of n-vertex tripartite graphs and characterize these tripartite graphs with nullity $n-4$. We also characterize some tripartite graphs with nullity $n-6$ in this class.

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 05C50
Key words and phrases: nullity, tripartite graph, expanded path

1 Introduction

Let G be a simple graph with vertex set $V(G) = \{v_1, \ldots, v_n\}$ and edge set $E(G)$. An edge joining a vertex v_i to a vertex v_j is denoted by v_iv_j (or v_jv_i). The *adjacency matrix* of G is $A(G) = [a_{ij}]_{n\times n}$, where

$$a_{ij} = \begin{cases} 1 & \text{if } v_iv_j \in E(G) \\ 0 & \text{otherwise} \end{cases} \quad (\forall v_i, v_j \in V(G)).$$

The eigenvalues of the graph G are the eigenvalues of $A(G)$ and the *spectrum* of G is the multiset of eigenvalues of G. The *nullity* of graph G, denoted by $\eta(G)$, is the multiplicity of the eigenvalue zero in the spectrum of G. The graph G is *singular* if $\eta(G) > 0$ and is *non-singular* if $\eta(G) = 0$. For graph theoretical terminology, we refer [10].

Collatz and Sinogowitz [2] posed the problem of characterizing singular graphs and since then, the theory of nullity of graphs has stimulated much research because of its importance in mathematics and chemistry. In literature, we find characterization of trees, unicyclic graphs, bicyclic graphs and bipartite graphs with their nullity.

Fiorini et al. [5] determined the greatest nullity among n-vertex trees in which no vertex has degree greater than a fixed positive integer D and gave a method of constructing the respective trees. Li and Chang [7] showed that there are some other trees with maximum nullity which can not be constructed by the method devised by Fiorini et al. [5] and modified the method of constructing the respective trees. Tan and Liu [11] found the nullity set of n-vertex unicyclic graphs with $n \geq 5$. They also characterized the unicyclic graphs with maximal nullity. The unicyclic graphs with minimum nullity are characterized by Li and Chang [8]. Li et al. [9] found the nullity set of n-vertex bicyclic graphs and gave a characterization of the bicyclic graphs with maximum nullity. Hu et al. [6] also found the nullity set of n-vertex bicyclic graphs, $n \geq 6$. Furthermore, the authors also characterized bicyclic graphs with extremal nullity. In Fan et al. [3], the authors found the nullity of unicyclic signed graphs and characterized n-vertex unicyclic signed graphs. Fan and Qian [4] introduced nullity of n-vertex bipartite graphs and presented characterization of bipartite graphs with nullity.

We find that the nullity of tripartite graphs does not follow as the extension of that of bipartite graphs. Thus it became interesting to characterize family of tripartite graphs with nullity. In this paper, we consider a class of n-vertex tripartite graphs, $n \geq 3$ and give nullity set of these tripartite graphs and characterize them with nullity $n-4$, $n \geq 4$. Furthermore, we discuss some tripartite graphs with nullity $n-6$, $n \geq 6$.

2 Preliminaries

In this section, we give some definitions, terminologies and known results which will be used later. Let G be an n-vertex graph with vertex set $V(G)$ and edge set $E(G)$. For $S \subseteq V(G)$, the *neighbor set* of S in G, denoted as $N(S)$, is a set containing those vertices of G that are adjacent to some vertex in S. If $S = \{v\}$, we write $N(v)$ for $N(\{v\})$. If S is nonempty, we denote by $G[S]$ the *induced subgraph* of G. The *rank* of the graph G, denoted by $\mathtt{rank}(G)$, is the rank of its adjacency matrix $A(G)$, that is, $\mathtt{rank}(G) = \mathtt{rank}(A(G))$. It is known that $\eta(G) = n - \mathtt{rank}(G)$. The graph G is said to be an *expanded graph* if its vertex set $V(G)$ can be partitioned into $V_1, V_2 \ldots, V_k$, $k \geq 2$, such that $G[V_i]$ is an empty graph, for $1 \leq i \leq k$. If $G[V_i \cup V_j]$ is a nonempty graph, it is a complete bipartite graph for $1 \leq i, j \leq k$, $i \neq j$. If G is an expanded graph on $V_1, V_2, \ldots, V_k$, each V_i, for $1 \leq i \leq k$, is called an *expanded vertex* of order $|V_i|$. We observe that each simple graph can be viewed as an expanded graph.

The n-vertex graph G is said to be an *expanded path* of length k if its vertex set $V(G)$ can be partitioned into $V_1, \ldots, V_k$, $k \geq 2$, such that

(i) $G[V_i]$ is an empty graph for $1 \leq i \leq k$,

(ii) $G[V_i \cup V_{i+1}]$ is a complete bipartite graph for $1 \leq i \leq k-1$,

(iii) $G[V_i \cup V_j]$ is an empty graph for $1 \leq i, j \leq k$ with $j \neq i+1$.

We use the notation $\mathbb{P}_k(V_1, \ldots, V_k)$ to denote an expanded path on $V_1, \ldots, V_k$ of length k. Similarly, an *expanded cycle* of length k, denoted by $\mathbb{C}_k(V_1, \ldots, V_k)$, is obtained from the expanded path $\mathbb{P}_k(V_1, \ldots, V_k)$ by adding edges between each vertex of V_1 and each of V_k. When there is no ambiguity, we simply write $\mathbb{P}_k$ and $\mathbb{C}_k$ respectively to represent an expanded path and an expanded cycle of length k. An *expanded decomposition* of the graph G is a list of expanded subgraphs such that each edge of G appears in exactly one expanded subgraph in the list.

The graph G is *tripartite* if its vertex set can be partitioned into three subsets X, Y and Z such that $G[X]$, $G[Y]$ and $G[Z]$ are empty graphs; such a partition (X, Y, Z) is called a *tripartition*. For any $S \subseteq V(G)$, we denote by $N_X(S)$ the neighbors of S in X. Analogously, we can define $N_Y(S)$ and $N_Z(S)$. We consider a special class of tripartite graphs defined as follows. Let $\mathcal{T}_n$ be the family of those n-vertex tripartite graphs G, $n \geq 5$, whose tripartition (X, Y, Z) satisfies the following:

$$N_X(Y') \neq X \text{ and } N_Z(Y') \neq Z \quad \forall \quad Y' \subseteq Y, \tag{1}$$

$$G[X \cup Z] \text{ is complete bipartite.} \tag{2}$$

Lemma 1 (Fan [4]) *An expanded path of length* $k \geq 2$ *and order* n *has nullity* $n-k$*, if* k *is even and* $n-k+1$*, if* k *is odd.*

Let K_{n_1,n_2} denote the complete bipartite graph, where n_1 and n_2 are the sizes of its partite sets. Also, K_1 denotes an isolated vertex and $r\,K_1$ denotes r copies of K_1. Then the following result characterizes the graphs with nullity $n-2$.

Lemma 2 (Cheng [1]) *Let* G *be an* n*-vertex simple graph,* $n \geq 2$*. Then* $\eta(G) = n-2$ *if and only if* $G \cong K_{n_1,n_2} \cup n_3K_1$*, where* $n = n_1 + n_2 + n_3$*,* $n_1, n_2 > 0$ *and* $n_3 \geq 0$*.*

3 Main results

Let $G \in \mathcal{T}_n$ with tripartition (X, Y, Z). The adjacency matrix $A(G)$ of G is defined by

$$A(G) = \begin{array}{c} \\ X \\ Z \\ Y \end{array} \begin{array}{c} \begin{array}{ccc} X & Z & Y \end{array} \\ \begin{bmatrix} \mathbf{0} & J & C_1 \\ J^t & \mathbf{0} & C_2 \\ C_1^t & C_2^t & \mathbf{0} \end{bmatrix} \end{array},$$

where J and $\mathbf{0}$ respectively denote the matrices with all entries 1 and 0. Let C and B denote the matrices defined as follows.

$$C = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} \mathbf{0} & J \\ J^t & \mathbf{0} \end{bmatrix}.$$

The matrix $A(G)$ can be viewed as

$$A(G) = \begin{bmatrix} B & C \\ C^t & \mathbf{0} \end{bmatrix}. \tag{3}$$

Let

$$U = [B \;\; C], \quad L = [C^t \;\; \mathbf{0}]. \tag{4}$$

Then $A(G)$ can be written as $A(G) = \begin{bmatrix} U \\ L \end{bmatrix}$.

For each $v \in X \cup Z$, we denote by U_v the row of $A(G)$ corresponding to the vertex v. Similarly, for each $v \in Y$, the row of $A(G)$ corresponding to the

vertex v is denoted by L_v. Let $S \subseteq X \cup Z$. Then from the matrix $A(G)$, we see that

$$\sum_{v \in S} k_v U_v = [b_1 \ b_2 \ c], \tag{5}$$

where b_1, b_2 are constant row matrices respectively of dimension $1 \times |X|$ and $1 \times |Z|$, c is row vector of dimension $1 \times |Y|$, and k_v's are real constants. Similarly, for any $Y' \subseteq Y$, we can write

$$\sum_{v \in Y'} k_v' L_v = [c_1 \ c_2 \ \mathbf{0}], \tag{6}$$

where c_1, c_2 and $\mathbf{0}$ are row vectors respectively of dimension $1 \times |X|$, $1 \times |Z|$ and $1 \times |Y|$, and k_v''s are real constants.

The following result gives information about the rank of a tripartite graph in $\mathcal{T}_n$.

Lemma 3 *Let $G \in \mathcal{T}_n$ with tripartition (X, Y, Z) and adjacency matrix defined by (3). Then*

$$\operatorname{rank}(G) = \operatorname{rank}(U) + \operatorname{rank}(L), \tag{7}$$

where U and L are defined by (4).

Proof. Let S and Y' be arbitrary subsets, respectively of $X \cup Z$ and Y. To prove (7), it is enough to show that $\sum_{v \in S} k_v U_v \neq \sum_{v \in Y'} k_v' L_v$ whenever $\sum_{v \in S} k_v U_v \neq 0$ and $\sum_{v \in Y'} k_v' L_v \neq 0$, and k_v's and k_v''s are real constants.

We can write $\sum_{v \in S} k_v U_v = [b_1 \ b_2 \ c]$ and $\sum_{v \in Y'} k_v' L_v = [c_1 \ c_2 \ \mathbf{0}]$, where b_1, b_2, c, c_1, c_2 and $\mathbf{0}$ are defined in (5) and (6). By condition (1), there exists a vertex in X which is not adjacent to any vertex in Y. Similarly, there exists a vertex in Z which is not adjacent to any vertex in Y. Thus there are at least two zero columns in C^t corresponding to a vertex in X and to a vertex in Z. That is, there are zero entries in vectors c_1 and c_2. Now, if $\sum_{v \in S} k_v U_v = \sum_{v \in Y'} k_v' L_v$ then $[b_1 \ b_2 \ c] = [c_1 \ c_2 \ \mathbf{0}]$. As b_1 and b_2 are constant vectors, the vectors b_1, b_2, c, c_1, c_2 are all zero vectors. This completes the proof. □

Corollary 4 *Let $G \in \mathcal{T}_n$ with tripartition (X, Y, Z) and the adjacency matrix $A(G)$ defined by (3). Then $\operatorname{rank}(G) = 2(1 + \operatorname{rank}(C))$.*

Proof. By the construction of the matrix $A(G)$ and by the arguments used in Lemma 3, we see that $\operatorname{rank}(U) = \operatorname{rank}(B) + \operatorname{rank}(C)$. Since $\operatorname{rank}(B) = 2$ and $\operatorname{rank}(L) = \operatorname{rank}(C) = \operatorname{rank}(C^t)$, we get from (7) that $\operatorname{rank}(G) = 2(1 + \operatorname{rank}(C))$. □

Let $\mathbb{C}_k(\overline{e})$ denote an expanded cycle of length k with an expanded chord $\overline{e}$ joining two non-adjacent expanded vertices of the cycle $\mathbb{C}_k$. We have the following observation.

Lemma 5 *If* $G = \mathbb{C}_5(\overline{e}) \cup kK_1$ *is a graph of order* n *shown in Figure 1,* $k \geq 0$*, then* $G \in \mathcal{T}_n$ *and* $\eta(G) = n - 4$*.*

Proof. Let $X = X_1 \cup X'$, $Z = Z_1 \cup Z'$ and $Y = Y_1 \cup Y'$, where Y' is possibly empty. Then we see that the graph G is a tripartite graph with tripartition (X, Y, Z). Moreover, G satisfies (1) because $N_{X'}(Y) = \emptyset$ and $N_{Z'}(Y) = \emptyset$. Also, $G[X \cup Z] = \mathbb{P}(X, Z)$, that is, G satisfies (2). Thus $G \in \mathcal{T}_n$. Let $A(G)$ be the adjacency matrix of G defined by (3). By the construction of G, we see that all rows of C^t are identical and therefore $\mathrm{rank}(C) = 1$. By Corollary 4, we conclude that $\eta(G) = n - 4$. □

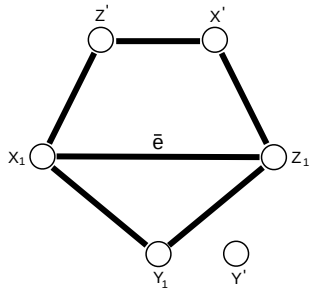

Figure 1: An expanded graph $\mathbb{C}_5(\overline{e}) \cup kK_1$

The next result gives the nullity set of the graphs in $\mathcal{T}_n$.

Theorem 6 *Let* m_1, m_2 *and* m_3 *be positive integers such that* $n = m_1 + m_2 + m_3$*. Then for each integer* $k \in \{0, 1, \ldots, \min\{m_1 + m_3 - 2, m_2\}\}$*, there is a graph* $G \in \mathcal{T}_n$ *with tripartition* (X, Y, Z) *such that* $|X| = m_1$*,* $|Y| = m_2$*,* $|Z| = m_3$ *and* $\eta(G) = n - 2(k + 1)$*. Conversely, if* $G \in \mathcal{T}_n$ *with tripartition* (X, Y, Z) *then* $\eta(G) = n - 2(1 + k)$*, where* $k \in \{0, 1, \ldots, \min\{|X| + |Z| - 2, |Y|\}\}$*.*

Proof. First, we prove that for each $k \in \{0, 1, \ldots, \min\{m_1 + m_3 - 2, m_2\}\}$, there is a graph $G \in \mathcal{T}_n$ with tripartition (X, Y, Z) such that $|X| = m_1$, $|Y| = m_2$, $|Z| = m_3$ and $\eta(G) = n - 2(k+1)$. We take three non-empty sets $X = \{x_1, x_2, \ldots, x_{m_1}\}$, $Y = \{y_1, y_2, \ldots, y_{m_2}\}$ and $Z = \{z_1, z_2, \ldots, z_{m_3}\}$. If $k = 0$, we construct a graph $G = P(X, Z) \cup m_2K_1$. Clearly, $G \in \mathcal{T}_n$. By using Lemma 2, $\eta(G) = n - 2$. If $k > 0$, we consider following two cases.

Case 1. When $k \leq m_1 - 1$. Since $k \leq m_2$, we construct a tripartite graph G with tripartition (X, Y, Z) that satisfies the following.

(i) $G[X \cup Z] = \mathbb{P}(X, Z)$,

(ii) $|N_X(y_i)| = 1$ and $N_Z(y_i) = \emptyset$ for $1 \leq i \leq k$,

(iii) $N_X(y_i) \cap N_X(y_j) = \emptyset$ for $i \neq j$ and $1 \leq i, j \leq k$,

(iv) $d(y_i) = 0$ for $k + 1 \leq i \leq m_2$.

Then $G \in \mathcal{T}_n$. Moreover, the adjacency matrix of G is given by (3), where

$$C = \left[\begin{array}{c|c} I_{k\times k} & \mathbf{0}_{k\times(m_2-k)} \\ \hline \mathbf{0}_{(m_1+m_3-k)\times k} & \mathbf{0}_{(m_1+m_3-k)\times(m_2-k)} \end{array}\right].$$

Then $\mathrm{rank}(C) = k$. By Corollary 4, we get $\eta(G) = n - 2(1 + k)$.

Case 2. When $k > m_1 - 1$. Since $k \leq m_2$ and $k - (m_1 - 1) \leq m_3 - 1$, we construct a tripartite graph G with tripartition (X, Y, Z) that satisfies the following.

(i) $G[X \cup Z] = \mathbb{P}(X, Z)$,

(ii) $|N_X(y_i)| = 1$ for $1 \leq i \leq m_1 - 1$,

(iii) $|N_Z(y_i)| = 1$ for $m_1 \leq i \leq k$,

(iv) $N_X(y_i) \cap N_X(y_j) = \emptyset$ for $i \neq j$ and $1 \leq i, j \leq m_1 - 1$,

(v) $N_Z(y_i) \cap N_Z(y_j) = \emptyset$ for $i \neq j$ and $m_1 \leq i, j \leq k$,

(vi) $d(y_i) = 0$ for $k + 1 \leq i \leq m_2$.

Then $G \in \mathcal{T}_n$. Moreover, the adjacency matrix of G is given by (3), where

$C =$

$$\left[\begin{array}{c|c|c} I_{(m_1-1)\times(m_1-1)} & \mathbf{0}_{(m_1-1)\times(k-m_1+1)} & \mathbf{0}_{(m_1-1)\times(m_2-k)} \\ \hline \mathbf{0}_{1\times(m_1-1)} & \mathbf{0}_{1\times(k-m_1+1)} & \mathbf{0}_{1\times(m_2-k)} \\ \hline \mathbf{0}_{(k-m_1+1)\times(m_1-1)} & I_{(k-m_1+1)\times(k-m_1+1)} & \mathbf{0}_{(k-m_1+1)\times(m_2-k)} \\ \hline \mathbf{0}_{(m_1+m_2-1-k)\times(m_1-1)} & \mathbf{0}_{(m_1+m_2-1-k)\times(k-m_1+1)} & \mathbf{0}_{(m_1+m_2-1-k)\times(m_2-k)} \end{array}\right].$$

Then $\mathrm{rank}(C) = k$. Corollary 4 gives $\eta(G) = n - 2(1 + k)$.

Conversely, we show that if $G \in \mathcal{T}_n$ with tripartition (X, Y, Z), then $\eta(G) = n-2(1+k)$ where $k \in \{0, 1, \ldots, \min\{|X|+|Z|-2, |Y|\}\}$. By Corollary 4, $\mathrm{rank}(G) = 2(1+\mathrm{rank}(C))$. By (1), there are at least two zero rows in C. This implies that $\mathrm{rank}(C) \leq \min\{|X| + |Z| - 2, |Y|\}$. The result is true by setting $\mathrm{rank}(C) = k$. □

From Corollary 4, for each graph $G \in \mathcal{T}_n$ with $A(G)$ defined by (3), we can write

$$\eta(G) = n - 2(1 + \mathrm{rank}(C)). \tag{8}$$

The next result is a direct consequence of Lemma 2.

Theorem 7 *For a graph $G \in \mathcal{T}_n$ with tripartition (X, Y, Z), $\eta(G) = n - 2$ if and only if $G = \mathbb{P}(X, Z) \cup |Y|K_1$.*

Proof. Let $G \in \mathcal{T}_n$ with tripartition (X, Y, Z) and $\eta(G) = n - 2$. Then from equation (8), we have $\mathrm{rank}(C) = 0$. That is, $d(y) = 0$ for all $y \in Y$. Thus $G = \mathbb{P}(X, Z) \cup |Y|K_1$. Conversely, suppose that $G = \mathbb{P}(X, Z) \cup |Y|K_1$. Using Lemma 1, we see that $\eta(G) = n - 2$. □

Theorem 8 *Let $G \in \mathcal{T}_n$ with tripartition (X, Y, Z) and $n \geq 4$. Then $\eta(G) = n - 4$ if and only if G is a graph H possibly with some isolated vertices, where H is an expanded path of length 4 or the expanded graph $\mathbb{C}_5(\overline{e})$.*

Proof. Let $G \in \mathcal{T}_n$ with tripartition (X, Y, Z) and $\eta(G) = n - 4$. Let $A(G)$ be the adjacency matrix of G defined by (3). Then by (8), we have $\mathrm{rank}(C) = 1$, that is, $\mathrm{rank}(L) = 1$. This means that there is only one independent row, say, L_{y_0} in L, where $y_0 \in Y$ is the vertex corresponding to L_{y_0}. Then for each $y \in Y$, either $N(y) = N(y_0)$ or $N(y) = \emptyset$. Let $Y_1 \subseteq Y$ is the set of all vertices of Y with non-zero degree. We have the following three cases.

Case 1. When $N_Z(Y_1) = \emptyset$. In this case, $N(Y_1) \subseteq X$. By condition (1), $N(Y_1) \neq X$. We partition X, Y and Z into Y_1, $N(Y_1)$, Z and $X \setminus N(Y_1)$. Then G can be drawn as an expanded path $\mathbb{P}(Y_1, N(Y_1), Z, X \setminus N(Y_1))$ possibly with some isolated vertices in $Y \setminus Y_1$.

Case 2. When $N_X(Y_1) = \emptyset$. In this case, $N(Y_1) \subseteq Z$. By condition (1), $N(Y_1) \neq Z$. We partition X, Y and Z into Y_1, $N(Y_1)$, X and $Z \setminus N(Y_1)$. Then G can be drawn as an expanded path $\mathbb{P}(Y_1, N(Y_1), X, Z \setminus N(Y_1))$ possibly with some isolated vertices in $Y \setminus Y_1$.

Case 3. When $N_X(Y_1) \neq \emptyset$ and $N_Z(Y_1) \neq \emptyset$. We can partition X into X_1 and X', such that $X_1 = N_X(Y_1)$ and $X' = X \setminus X_1$. Similarly, we can partition Z into Z_1 and Z', such that $Z_1 = N_Z(Y_1)$ and $Z' = Z \setminus Z_1$. Then, using condition (2),

one can draw the graph G as an expanded graph $\mathbb{C}_5(\overline{e})$ (shown in Figure 1). There are possibly some isolated vertices in Y', where $Y' = Y \setminus Y_1$.

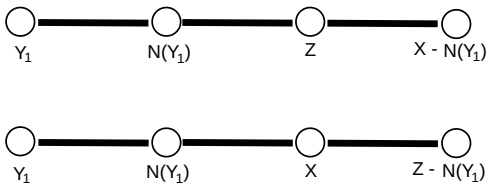

Figure 2: Two expanded paths $\mathbb{P}(Y_1, N(Y_1), Z, X \setminus N(Y_1))$ and $\mathbb{P}(Y_1, N(Y_1), X, Z \setminus N(Y_1))$ of length 4

Conversely, let G be drawn as an expanded path of length 4 possibly with some isolated vertices. Then Lemma 1 yields that $\eta(G) = n-4$. Furthermore, if G can be drawn as $\mathbb{C}_5(\overline{e})$ with some isolated vertices, then using Lemma 5, we get $\eta(G) = n-4$. □

4 Some graphs in $\mathcal{T}_n$ with nullity $n-6$

In this section, we consider some graphs in $\mathcal{T}_n$, $n \geq 6$, with nullity $n-6$. Let $G \in \mathcal{T}_n$ with tripartition (X, Y, Z) and let $X' = X \setminus N_X(Y)$. Note that $X' \neq \emptyset$ by (1). We assume that

$$G[N_X(Y) \cup Y] = \mathbb{P}(N_X(Y), Y). \tag{9}$$

The following result gives a characterization of a graph G in $\mathcal{T}_n$, $n \geq 6$ satisfying (9) and $\eta(G) = n-6$.

Theorem 9 *Let $G \in \mathcal{T}_n$ with tripartition (X, Y, Z), $n \geq 6$. Assume that G satisfies (9) and $\eta(G) = n-6$. Then G has one of the following expanded decomposition.*

(1) $\mathbb{C}_5(\overline{e})$, $\mathbb{P}_2$,

(2) $\mathbb{C}_5(\overline{e})$, $\mathbb{C}_3$, $\mathbb{P}_2$,

(3) $2\mathbb{C}_5(\overline{e})$, $2\mathbb{P}_2$,

(4) $\mathbb{C}_5(\overline{e})$, $\mathbb{C}_3$, $2\mathbb{P}_2$.

Proof. Let $G \in \mathcal{T}_n$ with tripartition (X, Y, Z) satisfying (9) and $\eta(G) = n - 6$, $n \geq 6$. Let $X' = X \setminus N_X(Y)$ and $Z' = Z \setminus N_Z(Y)$. From (1), we see that X' and Z' are nonempty. Let $A(G)$ be the adjacency matrix of G defined by (3). Since $\eta(G) = n - 6$, using Corollary 4, we have $\mathtt{rank}(L) = 2$. This implies that L has two independent rows, say, L_{y_1} and L_{y_2}, where $y_1, y_2 \in Y$. Using (1) and (9), the columns of L corresponding to the vertices of X are constant. Then for each $y \in Y$, either $L_y = L_{y_1}$ or $L_y = L_{y_2}$. Thus we partition Y into Y_1 and Y_2, where

$$Y_1 = \{y \in Y \mid L_y = L_{y_1}\}, Y_2 = \{y \in Y \mid L_y = L_{y_2}\}.$$

Note that $N_Z(y) = N_Z(Y_1)$ for each $y \in Y_1$, and $N_Z(y) = N_Z(Y_2)$ for each $y \in Y_2$. Since $\mathtt{rank}(L) = 2$, either $N_Z(Y_1) \neq \emptyset$ or $N_Z(Y_2) \neq \emptyset$. Without loss of generality, assume that $N_Z(Y_1) \neq \emptyset$ and $N_Z(Y_1) \not\subseteq N_Z(Y_2)$. The following three cases are possible.

Case 1. When $N_Z(Y_1) \cap N_Z(Y_2) = \emptyset$.
If $N_Z(Y_2) = \emptyset$, then $N_Z(Y) = N_Z(Y_1)$. We draw the graph G as an expanded graph on six expanded vertices $X_1 = N_X(Y)$, X', $Z_1 = N_Z(Y)$, Z', Y_1 and Y_2. Here Y' is possibly empty. The graph is shown in Figure 3 (i). In this case, we can decompose the graph G into $\mathbb{C}_5(\overline{e})$ and $\mathbb{P}_2$.

If $N_Z(Y_2) \neq \emptyset$, we partition $N_Z(Y)$ into $Z_1 = N_Z(Y_1)$ and $Z_2 = N_Z(Y_2)$. We draw the graph G as an expanded graph on seven expanded vertices $X_1 = N_X(Y)$, X', $Z_1 = N_Z(Y_1)$, $Z_2 = N_Z(Y_2)$, Z', Y_1 and Y_2. The graph is shown in Figure 3 (ii). In this case, the graph can be decomposed into $\mathbb{C}_5(\overline{e})$, $\mathbb{C}_3$ and $\mathbb{P}_2$.

Case 2. When $N_Z(Y_1) \cap N_Z(Y_2) \neq \emptyset$ and $N_Z(Y_2) \not\subseteq N_Z(Y_1)$.
Let $Z_1 = N_Z(Y_1) \cap N_Z(Y_2)$, $Z_2 = N_Z(Y_1) \setminus N_Z(Y_2)$ and $Z_3 = N_Z(Y_2) \setminus N_Z(Y_1)$. Then Z', Z_1, Z_2 and Z_3 form a partition of Z. The graph can be drawn as an expanded graph on eight expanded vertices $X_1 = N_X(Y)$, X', Z_1, Z_2, Z_3, Z', Y_1 and Y_2. The graph is shown in Figure 4 (i). In this case, the graph G can be decomposed into $2\mathbb{C}_5(\overline{e})$ and $2\mathbb{P}_2$.

Case 3. When $N_Z(Y_1) \cap N_Z(Y_2) \neq \emptyset$ and $N_Z(Y_2) \subseteq N_Z(Y_1)$.
We draw the graph G as an expanded graph on seven expanded vertices $X_1 = N_X(Y)$, X', $Z_1 = N_Z(Y_2)$, $Z_2 = N_Z(Y_1) \setminus N_Z(Y_2)$, Z', Y_1 and Y_2. The graph is shown in Figure 4 (ii). The graph G can be decomposed into $\mathbb{C}_5(\overline{e})$, $\mathbb{C}_3$ and $2\mathbb{P}_2$. □

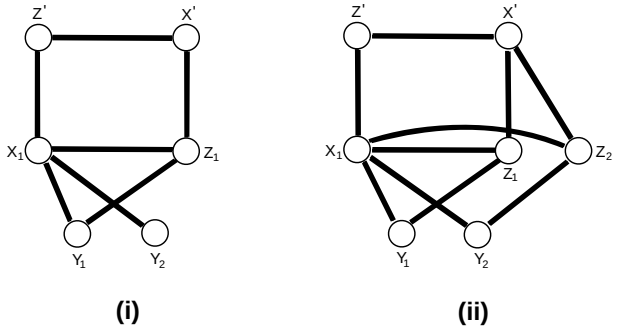

Figure 3: Graphs drawn in Case 1 with $N(Y_1) \cap N(Y_2) = \emptyset$

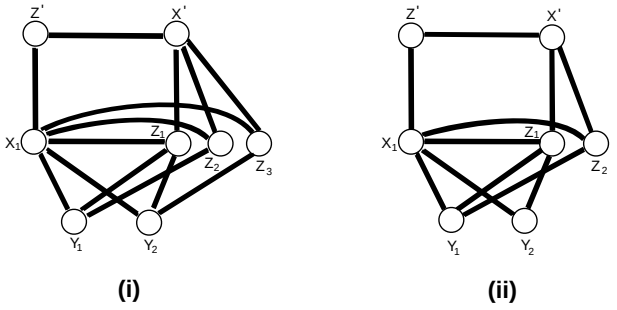

Figure 4: Graphs drawn in Case 2 and Case 3 with $N(Y_1) \cap N(Y_2) \neq \emptyset$

5 Conclusion

We studied n-vertex tripartite graphs satisfying (1) and (2). We obtained the nullity set of this class of n-vertex tripartite graphs and characterized them with nullity $n-4$. It will be interesting to consider a more general class of n-vertex tripartite graphs and to characterize them with their nullity. In Theorem 9, we characterized those n-vertex tripartite graphs whose nullity is $n-6$ and that satisfy (1), (2) and (9). We are not sure about the converse of Theorem 9 and it is left as an open problem.

References

[1] B. Cheng, B. Liu, On the nullity of graphs, *El. J. Lin. Algebra* **16** (2007) 60–67. ⇒99

[2] L. Collatz, U. Sinogowitz, Spektren endlicher grafen, *Abh. Math. Sere. Univ. Hamburg* **21** (1957) 63–77. ⇒97

[3] Y.-Z. Fan, Yue Wang, Yi Wang, A note on the nullity of unicyclic signed graphs, *Linear Algebra Appl.* **438** (2013) 1193-1200. ⇒97

[4] Y.-Z. Fan, K.-S. Qian, On the nullity of bipartite graphs, *Linear Algebra Appl.* **430** (2009) 2943–2949. ⇒97, 99

[5] S. Fiorini, I.Gutman, I. Sciriha, Trees with maximum nullity, *Linear Algebra Appl.* **397** (2005) 245–251. ⇒97

[6] S. Hu, T. Xuezhong, B. Liu, On the nullity of bicyclic graphs, *Linear Algebra Appl.* **429** (2008) 1387–1391. ⇒97

[7] W. Li, A. Chang, On the trees with maximum nullity, *MATCH Commun. Math. Comput. Chem.* **56,** 3 (2006) 501-508. ⇒97

[8] W. Li, A. Chang, Describing the nonsingular unicyclic graph, *J. Math. Study* **4** (2007) 442–445. ⇒97

[9] J. Li, A. Chang, W.C. Shiu, On the nullity of bicyclic graphs, *MATCH Commun. Math. Comput. Chem.* **60,** 1 (2008) 21–36. ⇒97

[10] S. Pirzada, *An Introduction to Graph Theory*, Universities Press, Orient Black-Swan, Hyderabad, India, 2005. ⇒97

[11] T. Xuezhong, B. Liu, On the nullity of unicyclic graphs, *Linear Algebra Appl.* **408** (2005) 212–220. ⇒97

Received: April 12, 2016 • Revised: May 30, 2016

Acta Universitatis Sapientiae

The scientific journal of Sapientia Hungarian University of Transylvania publishes original papers and surveys in several areas of sciences written in English.
Information about each series can be found at
http://www.acta.sapientia.ro.

Acta Universitatis Sapientiae, Informatica

Each volume contains two issues.

Sapientia University

DE GRUYTER OPEN

Scientia Publishing House

ISSN 1844-6086
http://www.acta.sapientia.ro

Information for authors

Acta Universitatis Sapientiae, Informatica publishes original papers and surveys in various fields of Computer Science. All papers are peer-reviewed.

Papers published in current and previous volumes can be found in Portable Document Format (pdf) form at the address: http://www.acta.sapientia.ro.

The submitted papers should not be considered for publication by other journals. The corresponding author is responsible for obtaining the permission of coauthors and of the authorities of institutes, if needed, for publication, the Editorial Board is disclaiming any responsibility.

Submission must be made by email (acta-inf@acta.sapientia.ro) only, using the LaTeX style and sample file at the address http://www.acta.sapientia.ro. Beside the LaTeX source a pdf format of the paper is necessary too.

Prepare your paper carefully, including keywords, ACM Computing Classification System codes (http://www.acm.org/about/class/1998) and AMS Mathematics Subject Classification codes (http://www.ams.org/msc/).

References should be listed alphabetically based on the Intructions for Authors given at the address http://www.acta.sapientia.ro.

Illustrations should be given in Encapsulated Postscript (eps) format.

One issue is offered each author free of charge. No reprints will be available.

Contact address and subscription:
Acta Universitatis Sapientiae, Informatica
RO 400112 Cluj-Napoca
Str. Matei Corvin nr. 4.
Email: acta-inf@acta.sapientia.ro

Printed by Idea Printing House
Director: Péter Nagy

ISSN 1844-6086
http://www.acta.sapientia.ro