Software visualization for predicting team elimination in league sports competitions.

A. Zec*
* University of Sarajevo, Faculty of Electrical and Electronic Engineering/ Computer Science and Informatics Department, Sarajevo, Bosnia and Herzegovina
amerzec@gmail.com

***Abstract* — This paper presents research on development of visualization software for modeling graph problems. In broader sense, major assignment of this work is using simple mathematical problem from graph theory to show the principles of modeling math problem with the help of software. Problem being modeled is prediction of team elimination in league sports competitions. This type of problem can be modeled as finding maximum flow in transport networks and can be classified as combinatorial optimization . Given model will be idealized in some aspects, but at the end, it won't stop us to show the power of computing when it is used as mathematical instrument.**

***Keywords* – Combinatorial optimization, Transport networks, Team elimination, Computer visualization, Graph algorithms applications.**

I. INTRODUCTION

Fans of professional sport teams are well known for their insatiable need of information about performances and possibilities of their loved teams. In order to provide simpler examples, we will take baseball in consideration as "league sport" in the rest of the paper. Of most importance (for fans) are odds of their team reaching playoffs. Many fans usually check papers and web-sites daily in search for their team progress. What different media report are usually one or another kind of statistical measures which they use to help fans in creating early picture of playoffs. Although being informative these statistics are created by pairing teams but ignore remaining games schedule. Given the standings in a sports league at some point during the season, we would like to determine which teams have been "mathematically eliminated" from winning their division. A team is eliminated if it cannot possibly finish the season in first place or tied for first place. The goal is to determine exactly which teams are eliminated. The problem is not as easy as many sports writers would have you believe, in part because the answer depends not only on the number of games won and left to play, but also on the schedule of remaining games. In another words, team is eliminated if it doesn't have enough games left to play to reach the number of wins that first-positioned team holds; so even if the left-behind team wins all games left, and first-placed never wins again, left-behind team can't do better than second place. Traditional method for eliminated teams determination is too conservative. Shortage of this method is simple to explain; it ignores important fact that every time first placed team loses the game, some other team has to win. If the first placed team loses all remaining games, then it "gives" enough wins to some other team (or teams) in a way that this team can end up with better result than initially observed left-behind team. In this case, left-behind team can't reach first place even if it wins all games left.

It turns out that simple max-flow calculation in small transport network can precisely determine when some team is eliminated. Being mentioned in [1], this elimination problem has long history in optimization literature. Schwartz [2] might have been the first who observed that team can be eliminated from first place early and posted first formulation of network to determine elimination to the closest date. Hoffman and Rivlin[3] extended this work to show necessary and sufficient conditions for elimination from *k*-th position. However, McCormick showed that this more general problem of determination of elimination from *k*-th position is NP-complete. The first part of this work paper provides more detailed presentation of problem. Second part classifies problem being solved and presents possible solution using transport networks. In this section we also focus on algorithms that are mission-critical for solving the problem. More precisely, there will be talk about Ford-Fulkerson algorithm and its well known optimization Edmonds-Karp algorithm in applied domain. Finally, in third section we demonstrate visualization software created for solving this mathematical problem, which could be of interest to those trying to teach graph theory, or anyone interested in computer visualization. One could use it for "step by step" demonstration of algorithms run. This section also devotes attention to few software design patterns used through development.

II. MATHEMATICAL BACKGROUND OF THE PROBLEM

A. Several complex examples

In baseball problem of elimination, we have a league of N teams. At some moment in season, team *i* has *w*[*i*] wins and *g*[*i*] [j] games left to play against team *j*. Team is eliminated if it can't finish season at first place or share first place with another team. Our problem is finding all which exact teams are eliminated. Answer depends not

only on number of games won and left to play, but also depends on schedule of games left to play. To see some complications let's consider following scenario:

TABLE I.
SIMPLE GAMES STATE

Team	Win	Lost	Left	**Atl**	**Phi**	**N.Y.**	**Mon**
Atl.	83	71	8	-	1	6	1
Phi.	80	79	3	1	-	0	2
N.Y.	78	78	6	6	0	-	0
Mon	77	82	3	1	2	0	-

Montreal is eliminated because it can end up with 80 wins at most, while Atlanta already has 83 wins. This is simplest case of elimination. However, there could be more complicated reasons. For example, Philadelphia is also mathematically eliminated. It can finish season with 83 wins at most, what looks enough to tie with Atlanta. But this would require Atlanta to lose all other left games, including 6 against New York, which gives New York 84 wins in total. This also means that New York is still not mathematically eliminated despite the fact that it has less wins then Philadelphia. It is not always easy for sports journalists to explain why is some team eliminated. Here we provide another example:

TABLE II.
MORE COMPLICATED GAMES STATE

Team	Win	Loss	Left	**N.Y**	**Bal**	**Bos**	**Tor**	**Det**
N.Y.	75	59	28	0	3	8	7	3
Bal	71	63	28	3	0	2	7	4
Bos	69	66	27	8	2	0	0	0
Tor	63	72	27	7	7	0	0	0
Det	49	86	27	3	4	0	0	0

At first we can see that total number of games left to play is not necessarily equal to number of games left to play against rivals in same division as some teams can play opponents outside of their own division. By looking at TABLE II. one could think that Detroit still has chances to catch New York and win division because Detroit can end with 76 wins at most. But following observation assures that this is not the case and gives argument of Detroit being eliminated.

By winning all left games, Detroit can end season with 76 wins and 86 loses. If New York wins at least 2 more games, they will end season with record of 77 wins and 85 looses and it brings them ahead of Detroit. Therefore let's take a look at what happens if Detroit goes without any looses in the rest of the season, and New York has bad luck and doesn't win any more games.

The problem with this scenario is in New York which still has 8 games against Boston. If Boston wins all these games, they will end up season with at least 77 wins and it will bring them ahead of Detroit. Because of this, the only way for Detroit to finish at first place would be letting New York to win exactly one of 8 games with Boston and loose all other games against Boston. Besides that, Boston has to loose all games it plays with other teams except New York. So in the end this puts them in three-tie position as showed in TABLE III.

Now, let's take Baltimore and Toronto into consideration in our scenario. Baltimore has 2 games left with Boston and 3 games with N.Y. If everything happens as described, Baltimore can have at least 76 wins. What this means for Detroit is that they can catch up Baltimore only if it loses all games with other teams except New York and Boston. Actually, this implies Baltimore has to lose all 7 left games with Toronto.

TABLE III.
TRIPLE TIE SITUATION

Team	Wins	Loss
N. Y.	76	86
Bal	?	?
Boston	76	86
Toronto	??	??
Detroit	76	86

To make things more complicated, Toronto also has 7 games left to play with New York and we have already said in the beginning that Toronto has to win all these (i.e. New York has to give away these 7 games). Finally if this comes true, Toronto wins at least 14 more games what gives them final record of 77 wins and 85 looses or better, and they end up in front of Detroit. As we can see, whatever happens in the season from this moment, Detroit can't have first position in the league.

B. Graph theory solution to the problem

In this section we will show one possible solution of our problem by reducing it to maximum flow problem in transport networks for which the software was developed. To be able to check weather some team x is eliminated or not, we create network and solve max. flow problem in it. We have vertices that represent teams (all teams except team x whose elimination we are checking) and vertices that represent left games in division (all but games that include our team x). Intuitively every unit of flow in network corresponds to game left to play. We connect artificial vertex s (source) with every vertex of game i-j and put capacity of that edge to $g[\,i\,][\,j\,]$ (number of left games team i has to play against team j). If flow uses all $g[\,i\,][\,j\,]$ units of capacity on this edge, we think about these games as already played and wins are distributed in between vertices of team i and team j. Every vertex of game i-j is then connected with two opposed teams to make sure one team wins. On these edges we do not need constraints on amount of flow. Finally, we connect every team-vertex to artificial vertex t (drain). Team x can win at most $w[\,x\,] + r[\,x\,]$ games (where $r[\,x\,]$ is number of team x games left against other teams). In order to prevent team i to win more games than $(w[\,x\,] + r[\,x\,])$ in total, we include edge from team-vertex i on drain with capacity $w[\,x\,] + r[\,x\,] - w[\,i\,]$.

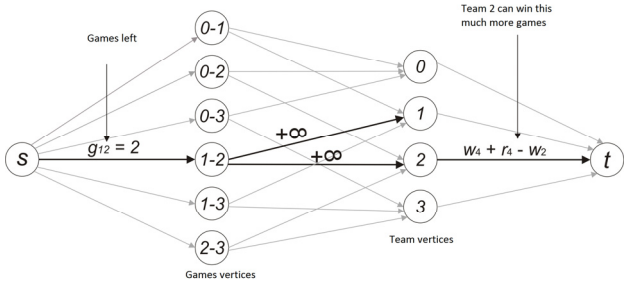

Figure I. Max flow graph model of team 4 elimination examination.

If the total flow in network shown in Figure I. satisfies all edges leaving *s* this means all winners are assigned to left games in a way that there is no team with more wins than team *x*. If maximum flow does not satisfy all edges leaving *s* , in that case there is no scenario in which team *x* wins division.

C. *Simplifying assumptions*

We make assumption there are no tied games. We also make assumption there are no canceled games, i.e. every scheduled game is played. Besides that we ignore joker possibilities when team can reach playoffs even if it doesn't end up being first in its own division.

D. *What max-cut tells us*

By solving max-cut problem, we can find out which teams are eliminated. It would be also nice if we could explain reason of team elimination to a friend, without touching graph theory. Here is once again more detailed and persuasive argument for Detroit elimination from previous example.

With all possible luck, Detroit can end season with 76 = 49 + 27 wins at most. Lets watch set of teams R={New York, Baltimore, Boston, Toronto}. Together they have 278 wins among them. There is total 27 = 3+8+7+2+7 games left to play among them, so collectively, this four teams have to win at least 27 more games. In average, teams from set R are winning (278+27)/4 = 305/4 = 76.25 games. As the number of games won has to be integer, some teams from R will have more then 76 and some less than 76 wins. Depending on the outcome, some team in R will have at least 77 wins, and because of that it becomes cause of Detroit's elimination.

Basically, whenever team is mathematically eliminated, there should be persuasive *elimination certificate,* even if the subset of teams in R doesn't need to contain teams from same division like in our example. However, we always can find subset R by choosing team-vertices on the "source" side of minimal *s-t* cut in given network. One could notice that by using max flow and max cut for finding R, once we find it, argument for team elimination does not include any sophisticated mathematics.

III. Software Visualization

For the purpose of solving max flow problem presented above, special software application was developed in Java language. Besides of solving given problem, software is capable of visualizing steps in max flow calculation. Program consists of several components which will be described in next few sections. At its core, it implements two well known algorithms from graph theory, Ford Fulkerson method and Edmonds Karp optimization.

A. *User input*

From the user perspective, it is possible to define all league parameters, including number of teams, their names and statistics. Complete user input is developed as wizard that leads user and validates final input data. Validation assumes that all statistical constraints are matching, i.e. number of games team *a* plays against team *b* equal to games team *b* has to play with *a* etc. Part of this wizard is given in Figure II.

Figure II. Wizard for games statistics.

B. *Network plotting and visualization*

After initial league data was conducted, program plots initial transport network as it was described in part II of this paper. Each vertex is given name. Team vertices have team name, and game vertices have name of teams playing the game. This way it is easier for user to see the movement of the flow when algorithms are running. Each edge also has label which gives us current flow and total capacity. For the purpose of plotting graphs very robust framework named JGraph[4] was used. It has capabilities of placing vertices and edges based on different layouts what makes final look and feel of graph very nice.

Whole program display consists of three docked windows. Main window contains graph and controls for choosing team for elimination and algorithm execution speed. This window is shown in the Figure III.

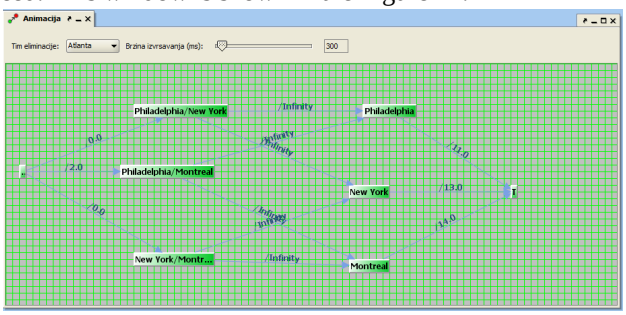

Figure III. Main animation window.

Second window is table of teams and games statistics. It can be helpful to user in analyzing results after visualization and algorithms are finished. It is actually the same table from wizard in Figure II made visible again.

Finally third window is user console that provides textual representation for each atomic step of algorithm execution and gives rational and simple explanation of why some team is eliminated or not. This is shown in Figure IV.

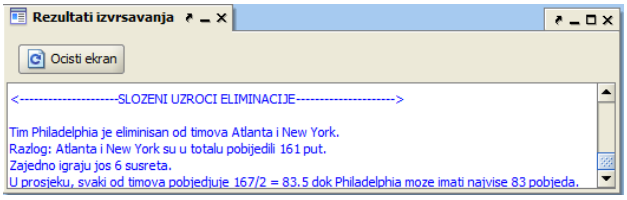

Figure IV. Console window.

After the user has decided which team elimination is questionable, it is possible to run execution with one of three controls available. These controls make possible for user to stop algorithm in execution and make analysis after each step. This can be very useful if someone faces Ford-Fulkerson or Edmonds-Karp algorithms for the first time, which makes learning curve not too steep.

Algorithm execution is divided in atomic steps. Every step being executed makes coloring of the graph and output to the console window. One step of animation is shown in the Figure V. Besides possibility of stopping visualization it is possible to change execution speed while animation is running.

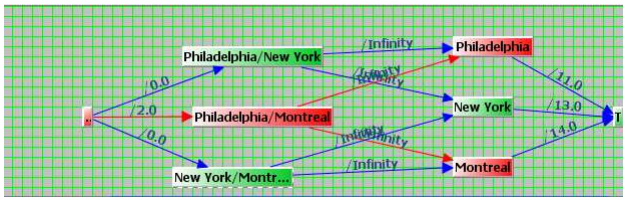

Figure V. Simple animation step.

C. Algorithms customization and design patterns

Ford-Fulkerson algorithm is very known for finding maximum flow in the graph. In the background it makes use of Depth-first-search for finding paths between *S* and *T* vertex. However, if we implement Breadth-first search then we get optimization called Edmonds-Karp algorithm. Both variations are used in our software and user can choose between any of them.

To be able to perform tricks on animation, algorithm implementations had to be filled with calls to different animation operations. It is known that graphics operations can be very costly and in this case it could damage algorithms performance. This brings us to problem of de-coupling algorithm execution from animation. Solution was based on command pattern. After each atomic operation (relevant for animation) is executed in algorithm, our algorithm creates object which is command for animation. But this commands are not executed until algorithm execution is over. Instead, we make use of queue (data structure) to fill it with all commands that need to be executed in order. Putting object in queue is constant operation *O(1)*, and therefore our algorithms don't lose any performance.

Another implementation challenge was how to make animation graphics to respond to commands being sent. For this purpose another well known design pattern was used, named Observer-Observable pattern. After we get initial queue of command that need to be performed on graph we register all nodes and edges as observers of animation being observable. Whenever event for certain node is directed to animation, it automatically knows what node to color. Edges coloring and labeling is done in similar fashion.

ACKNOWLEDGMENT

I am grateful for valuable suggestions from professor Željko Jurić, and my colleague Dino Kečo who encouraged me to bring my sports spirit into software development. Needless to say, any remaining errors are my own responsibility.

REFERENCES

[1] Ilan Adler, Alan L. Erera, Dorit S. Hochbaum, Eli V. Olinick: *Baseball, optimization, and the World Wide Web*, *Berkeley*, July 1998.

[2] Schwartz, B. L. 1966. *Possible winners in partially completed tournaments. SIAM Rev.* **8**(3) 302–308.

[3] Hoffman, A. J., T. J. Rivlin.: *When is a team 'mathematically' eliminated?* H. W. Kuhn, ed. *Proc.Princeton Sympos. on Math. Programming*. Princeton University Press, Princeton, NJ.,1970

[4] David Benson, *Jgraph and Jgraph Layout Pro User Manual*, *Northampton*, June 2009.