A Tabu Search Based Approach for Graph Layout

Fadi K. Dib
Computer Science Department
Gulf University for Science and Technology
Kuwait, Kuwait
deeb.f@gust.edu.kw

Peter Rodgers
School of Computing
University of Kent
Canterbury, UK
P.J.Rodgers@kent.ac.uk

***Abstract*—This paper describes an automated tabu search based method for drawing general simple graph layouts with straight lines. To our knowledge, this is the first time tabu methods have been applied to graph drawing. We formulated the task as a multi-criteria optimization problem with a number of metrics which are used in a weighted fitness function to measure the aesthetic quality of the graph layout. The main goal of this work is speeding up the graph layout process without sacrificing the layout quality. To achieve this we used a tabu search based method that goes through a predefined number of iterations to minimize the value of the fitness function. Tabu search always chooses the best solution in the neighborhood. This may lead to cycling, so a tabu list is used to store moves that are not permitted so that the algorithm does not choose previous solutions for a set period of time. We give experimental results applied on random graphs and we provide statistical evidence that our method outperforms one of the fast search-based drawing methods (hill climbing) in execution time while it produces comparably good graph layouts. We also demonstrate the method on real world graph datasets.**

Keywords—Information visualization; graph drawing; graph layout; tabu search

I. INTRODUCTION

In this work we address the research area of graph drawing. Here, the goal is to lay out a network diagram so that it can be analyzed and examined by users. There are several multi-criteria approaches to graph drawing which are based on explicit cost functions that combine several metrics of graph layout quality. This approach has the advantage of allowing explicit, tunable combinations of metrics to meet user preferences. However, such methods work slowly, typically taking a considerable time to lay out the graph. The contribution of this paper is to improve the performance of such systems by introducing tabu search features. To our knowledge, this is the first time tabu methods have been applied to drawing general simple graph layouts with straight lines.

Search based methods typically measure a number of metrics and combine them to form a fitness measure. When a new solution is found (perhaps by moving a node) the metrics are calculated again and a new fitness measure is found. This process happens a large number of times during the search, and so the process of finding a good layout is slow. Many drawers in the literature used search based methods, such as simulated annealing [2, 3], genetic algorithms [4, 5, 6, 7] and hill climbing [8, 9]. These produce good layouts, but they have great potential for improvement. For example, simulated annealing adds an element of non-determinism in order to escape from local minima in the search space. This slows down the performance of the algorithm since this stochastic behavior means that a larger number of iterations would be necessary to reach a minimum in the search space. Genetic algorithms, on the other hand, typically have an even slower rate of convergence compared to simulated annealing and hill climbing as it makes a wider search of the problem space. The main problem with hill climbing is that it gets trapped in local optima.

Our main goal in this work is concerned with improving the drawer's efficiency by speeding up the drawing process, using a search based method known as tabu search, without sacrificing the layout quality. We are not looking for the global optimum solution, but aim to obtain a good optimal solution quickly. Therefore, we compare our approach with hill climbing. In addition to its simple implementation, hill climbing has proven its efficiency in graph drawing applications [8, 9]. The main disadvantage of hill climbing is the likelihood of finding a sub-optimal local minimum in the search space. However, as the method is completely deterministic, comparison against hill climbing is more reliable than against the non-deterministic approaches of simulated annealing and genetic algorithms.

Tabu search is a general search based technique proposed by Glover [10, 11, 12] for finding good solutions to combinatorial optimization problems. It is considered to be a neighborhood search method (like simulated annealing) but it takes a more aggressive approach. It proceeds on the assumption that there is little benefit in choosing an inferior solution unless it is necessary, as in the case of escaping from a local optimum [13]. In other words, tabu search improves the efficiency of exploration process by keeping track of local information (like the current value of the objective function) along with information related to the exploration process. This systematic use of memory is an essential property of this searching technique. Tabu search keeps information on the itinerary through the last solutions visited. The role of this is to restrict the choice of some subsets in the neighborhood by forbidding moves to some neighbor solutions that have already been visited [14]. This constrains the direction of the search process by preventing the algorithm from going back to a previously reached state. At each iteration of the exploration process, it selects the best solution in the neighborhood. This is

unlike hill-climbing as it might make a down-hill move. Therefore, this technique does not run out of choices for the next move. However, this might lead to cycling by trapping the algorithm at locally optimal solutions. This problem can be resolved by introducing two structures called *tabu lists* and *aspiration functions* which are used to keep information about past moves in order to respectively constrain and diversify the search for good solutions [13].

Tabu search has shown good results and comparably fast solutions for some graph theory applications such as graph partitioning [13], graph coloring [15] and straight line crossing minimization [16, 17, 18]. It has also been used to solve different multiple objective optimization problems. The algorithm was used to solve four different applications in different areas [19]. In every application, the solutions were at least as good as, if not better than, the reported results using different search based techniques. Tabu search has been applied to the problem of routing school buses [20]. This algorithm was shown to be competitive in a set of problem instances for which a scatter search method was applied. Other researchers [21] proposed a tabu search algorithm as meta-heuristic method for network reconfiguration of multiple objectives problems in a radial distribution system. The work concluded that tabu search can quite easily handle the complicated constraints that are typically found in real-life applications. However, it failed in some circumstances for the following two reasons: an insufficient understanding of fundamental concepts of the tabu search method; and a lack of understanding of the problem in hand.

Our paper describes an approach for drawing general simple graphs with straight lines. This is achieved with a tabu search based method which draws general graphs with multiple aesthetic criteria that include node-node occlusion, edges length, edge crossings, and angular resolution. These criteria are used in a weighted fitness function to measure the quality of the graph layout. Whilst there have been empirical studies of what may be the most effective criteria for layout [1], we are not overly concerned with the particular criteria or their weights: our method would work effectively with other criteria or weightings.

The method goes through predefined number of iterations to minimize the value of the fitness function and it uses a tabu list to store tabu moves in order to prevent the algorithm from choosing previously reached moves for particular nodes for a period of time. We have tested our method on random graphs of different sizes and we describe the experiments and statistical analysis that brought us to the conclusion that our tabu search based method produces graph layouts as good as, if not better than, layouts drawn with a hill climbing method with a clear improvement in the time spent to draw the graph. We have also recognized improvements in both quality and time over hill climbing when the method is applied to real world graphs.

The rest of this paper is organized as follows: Section II describes some background in using search based techniques in graph layout; Section III describes our approach; Section IV describes experimental results on random graphs; Section V describes the results of applying our approach to real world graph datasets; finally, Section VI gives our conclusions and suggests future work.

II. RELATED WORK IN GRAPH LAYOUT

Multi-criteria graph layout can be modelled as a multiple objective optimization problem. When an algorithm attempts to draw a graph layout according to several graph aesthetic criteria, some of these criteria might conflict with each other. Hence, a fitness function that linearly combines all criteria is formulated. The optimizer attempts to minimize this function.

The problem with using general fitness functions is that it is usually computationally expensive to find a minimum fitness value. Since the overall fitness function might include both continuous and discrete measures, general search based approaches, such as simulated annealing, genetic algorithms, and hill climbing, have been used in order to find a minimum fitness value [35, 36].

Simulated annealing was the first general search method to be applied to the graph layout problem [2]. It was used to draw undirected graphs with straight line edges, taking into account several drawing aesthetics: distributing nodes evenly, making edge lengths uniform, minimizing edge crossings, and placing nodes not too close to edges. All these criteria were combined into a function that could be subject to a general optimization fitness function. This search based approach models the physical process of heating a material and then slowly cooling the temperature to decrease defects, so minimizing the system energy. It is often used for large-scale combinatorial optimization problems and implemented in a way that tries to escape from local minimum to global minimum by applying uphill moves (moves that worsen, rather than improve, the temporary solution). This allows the approach to escape from some local minimal solutions but with no guarantee that a global minimum can be reached eventually. The algorithm produces nice graph layouts for small sized graphs. However, it does not perform well for larger graphs.

An adjustment to simulated annealing approach was made in the algorithm proposed in [3]. Here, the fitness function is minimized using gradient descent. The gradient vector of the fitness function represents the direction in which the node should move to increase the value of the fitness function. Thus, this algorithm will move the node to the opposite direction to minimize the value of the fitness function. But this method is still slow when being applied on large graphs and it has some challenges. For example, the fitness function needs to be expressed explicitly in terms of coordinates as its derivative must be found. Some criteria, such as minimizing edge crossings, are discontinuous and not differentiable.

Genetic algorithms have also been applied to the graph layout problem. A genetic algorithm approach for drawing graphs under a number of visual constraints was proposed in [4, 5]. The proposed algorithm produces graphs with good quality in addition to its flexibility. It can be easily adapted to take new layout aesthetics into account. However, the major problem in this algorithm is its slow rate of convergence. It initially makes rapid progress towards a solution, but then it

converges very slowly to a global optimum, or at least to a good local one.

A genetic algorithm with local fine tuning based on the spring algorithm for the drawing of undirected graphs with straight-line edges has been proposed in [6]. According to some preliminary results, the algorithm produces layouts with a minimal number of edge crossings on all test graphs. The algorithm benefits from the combination of the genetic algorithm and the spring algorithm to produce good layouts for a large class of graphs with implicit symmetry, similar spring lengths, and even distribution of nodes. Although the layouts found by this algorithm have good general structures, some fine tuning might still be needed. Moreover, the comparatively long running time of the algorithm is a key disadvantage. One reason for the high time complexity of the algorithm comes from the chosen crossover operator to solve the competing conventions problem which states that a recombination of two good parents may yield a very poor offspring.

Similar work was introduced in [7]. This proposed a genetic algorithm that nicely draws undirected graphs of moderate size. But the algorithm still suffers from the lack of proper crossover operation which would speed up the computations by decreasing the number of needed generations.

Hill climbing is another search based approach that has been used in the field of graph drawing. It is one of the simplest search based algorithms used in the field of artificial intelligence. It is good for finding local optimum but it is not guaranteed to find the global optimum out of all possible solutions. It works by iteratively improving a given solution, which is often selected in a random way, by applying a transformation in the current solution or picking any solution in its neighborhood. Then, the new solution is compared to the old one. If the new solution is better than the old one, the new solution substitutes the old one. This process is repeated until a maximum number of repetitions is reached.

Hill climbing has been used to minimize number of edge crossings [8]. The experiments conducted on random graphs of different sizes showed that stochastic hill climbing outperforms efficient and popular search based techniques such as evolution strategies and genetic algorithms.

A hill climbing approach has been used to implement an automatic mechanism for drawing metro maps [9]. Metro map drawing is a specialized form of graph drawing. A good metro map layout has evenly spaced stations, lines at regular angles (typically multiples of 45 degrees) and labels placed in unambiguous locations. This work applied multi-criteria optimization using a fitness function consisting of five different metrics in a weighted sum, along with some rules that prevented some bad moves for each station (e.g. a station that was north of another station could not be moved south of it). A hill climbing algorithm was used to reduce the fitness function and find improved map layouts. Since hill climbing does not guarantee finding the global minimum, a clustering technique was applied to the map. The hill climber moves both stations and clusters when finding improved layouts. The mechanism produces good map layouts and in some cases better than both published and distorted layouts. However, the performance of the algorithm is slow.

Search based methods used to solve graph layout problem are generally successful in producing graphs with nice layouts but just for small or mid-sized graphs. In addition, the execution time of these methods is very slow.

III. OUR APPROACH

This section describes the basic concepts of our approach. We detail the algorithm of tabu search drawer, the criteria measured, and the method for combining criteria to produce a fitness value.

In outline, as described in Algorithm 1, our tabu search method operates in the following manner: first, we find a random initial graph layout such that no two nodes have the same position. We compute the fitness value of the initial layout. Then the following steps are performed for a predefined number of iterations: for each node, we search the points around a square centered on the node at a given distance, as shown in Fig. 1. Eight points around the square are checked (above, below, left, right, and the four corners). The ratio of the current solution's fitness function value with the previous solution's fitness function value is computed at each point around the square. Solutions with fitness function ratios above or equal to a predefined threshold value (`tabuCutOff`), are considered as tabu moves and will be stored in a tabu list. We then move the node to a non-tabu point where the value of the fitness function is a minimum compared to all the points of the square even if the new point does not improve the current value of the fitness function (this is why tabu search does not run out of solutions) and the previous solution becomes a tabu solution. After an arbitrary chosen number of iterations, as a cooling down process, the square size centered around the node is reduced and the `tabuCutOff` value is decreased to intensify the searching process. Finally, the tabu list is updated by removing old solutions from the list after a number of iterations in which a move should remain in tabu list before it can be released (`tabuDuration`) in order to diversify the searching space. Fig. 2 presents two examples of graph layouts drawn by our approach.

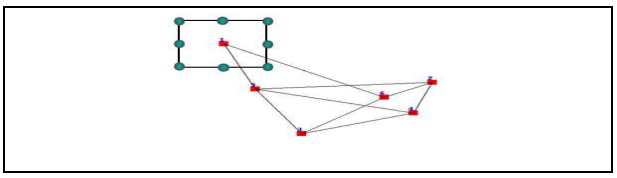

Fig. 1. Example of the points around the square checked by our algorithm on each node

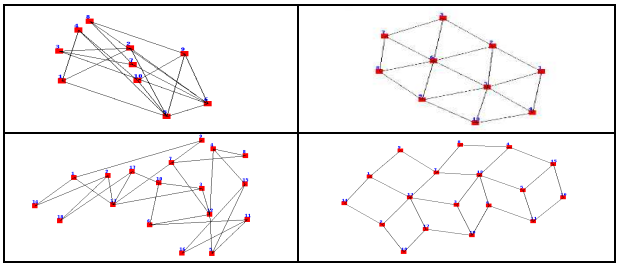

Fig. 2. Examples of layouts before (left) & after (right) applying our approach

Algorithm 1. Tabu Search Drawer

Given:

Connected Graph `G(V,E)`: `V` is a set of nodes and `E=(VxV)` is a set of edges.

`max_iterations`: predefined maximum number of iterations of the drawer.

`coolDown_iterations`: predefined number of iterations in which the process starts cooling down

`tabuDuration`: predefined number of iterations in which a move should remain in the tabu list.

`tabuCutOff`: predefined minimum value that determines whether a move is tabu or not.

Algorithm:

```
1: InitializeTabuList()
2: G_Layout = RandomizeLayout(G)
3: iterations = 0
4: while (iterations < max_iterations)
5:  for v ∈ V do
6:   for squarePos ∈ allNonTabuSquarePositions do
7:     currentPos = v_pos
8:     fitness_Current = Fitness(G_Layout)
9:     Update G_Layout s.t. v_pos = squarePos
10:    fitness_New = Fitness(G_Layout)
11:    if(fitness_New / fitness_Current > tabuCutOff)
12:     addToTabu(v,squarePos)
13:   end if
14:  end for
15:  v_pos = Min_NonTabuPosition()
16:  addToTabu(v,currentPos)
17: end for
18: if(iterations mod coolDown_iterations == 0)
19:  ReduceSquareSize()
20:  Decrease(tabuCutOff)
21: end if
22: if(iterations ≥ tabuDuration)
23:  RemoveTabuSolutions(iterations-tabuDuration)
24: end if
25: iterations = iterations + 1
26: end while
```

Our fitness function follows the standard approach for search based graph drawing methods. We implemented four metrics for measuring the quality of the graph [2, 9]. These represented the aesthetics of: distributing nodes evenly, making uniform edge lengths, minimizing edge crossings, and improving angular resolution. All these metrics contribute in the graph quality fitness function which is computed as follows:

$$fitness = w_1*m_1 + w_2*m_2 + w_3*m_3 + w_4*m_4 \quad (1)$$

where w_i and m_i are the weight and the measure for criteria i respectively. The problem in a multiple objective optimization function is that the value of a specific measure may dominate the others. Therefore, we applied a normalization process to ensure that the value of each measure is between 0 and 1.

We cannot determine unified weights that work properly for any graph. Therefore, the weights should be assigned by decision makers according to their preferences on which measure they prefer to dominate. We assigned the value 1 to all weights in order to avoid domination of one measure over another.

We realized that re-computing the fitness function at each point is a time consuming process. Therefore, we modified the way of computing the value of the function by caching the results such that the old value of the function is used to compute the new value. We just compute the change made in the function when a node is moved. When a node moves to a new position, the amount of change in the fitness function value, between the previous position of the node and its new position, is computed. If the change in the value improves the fitness function, we subtract the amount of the change made by the previous position of the node, and we add the amount of the change made by the new position of the node to get the new value of the fitness function. This process increased the speed of our method.

To tune the two parameters `tabuDuration` and `tabuCutOff`, we performed several experiments. We generated 100 random graphs (different to those shown in Table I and II). These were divided into 5 sets such that each set had different number of nodes and edges. Hence, each set consisted of 20 test cases with the same number of nodes and edges. The characteristics of the five sets were exactly the same of the first five groups of the graphs in the second category as will be described in the next section. We tested the drawer on the 100 test cases for six different values of `tabuDuration` {5, 6, 7, 8, 9, and 10} and for six different values of `tabuCutOff` {50, 60, 70, 80, 90, and 100}. In most of the cases, the best fitness function values and the shortest execution time were generated when the values of the two parameters were: `tabuDuration` = 7 and `tabuCutOff` = 80.

IV. EXPERIMENTAL RESULTS

The programming language used in our implementation is Java (version 1.7.0; Java HotSpot™ 64-Bit Server VM 21.0-b17 on Windows 7). We have tested our approach on different random graphs of different sizes. The experiments were performed using Lenovo Thinkpad T430, Intel® Core™ i7-3520M CPU processor with frequency of 2.90 GHz and 8 GB RAM.

We generated random graph datasets in two categories. The graphs of the first category have the same number of nodes but with different densities (i.e. different number of edges), whereas the graphs of the second category have different number of nodes with varying values of densities.

Our random graph generator generated random connected graphs. The parameters to it were the number of nodes and the density of the graph. It generated random locations for the nodes based on the size of the window where the graph will be displayed. Then, the generator chose random nodes as end points of edges. Self-sourcing edges and multiple edges between the same pair of nodes were not allowed. Finally, the generator tested the connectivity of the generated graph by running a breadth first search algorithm. Only connected graphs were accepted.

There were 200 random graphs in the first category split into 10 groups of 20 test cases each. All the graphs in this category had 150 nodes, randomly positioned. However each group had a differing number of edges than the other groups so that the density varied. However, the graphs in each group had same number of edges but with different initial layouts. See TABLE I for characteristics of the graphs in the first category.

The second category also had 200 random graphs, again split into 10 groups. Number of nodes in each group was increased by 50. The value of the density was chosen for each group to avoid too dense graphs. A similar random process used to generate graphs in the first category was applied to this category. See TABLE II for characteristics of the graphs in the second category.

TABLE I
Characteristics of the graphs in the 1st category

Group	Nodes	Edges	Density
1	150	558	0.05
2	150	1117	0.1
3	150	1676	0.15
4	150	2235	0.2
5	150	2793	0.25
6	150	3352	0.3
7	150	3911	0.35
8	150	4470	0.4
9	150	5028	0.45
10	150	5587	0.5

TABLE II
Characteristics of the graphs in the 2nd category

Group	Nodes	Edges	Density
1	50	153	0.125
2	100	544	0.11
3	150	1173	0.105
4	200	1890	0.095
5	250	2645	0.085
6	300	3363	0.075
7	350	3969	0.065
8	400	4788	0.06
9	450	5556	0.055
10	500	6237	0.05

We have applied our tabu search based approach and the hill climbing approach to the randomly generated graphs. All the weights of the metrics in the fitness function were equal. The metrics were normalized and therefore, equalizing the weights would equalize the effect of each metric on the value of the fitness function.

We note that the hill climbing approach used the same optimized fitness function calculations that the tabu search applied – only changes to the fitness function from moved nodes were recalculated.

To make a comprehensive comparison between tabu and hill climbing, we divided our experiments into three phases. In phase I, we applied both methods on the graphs of the two categories. The methods executed on the 20 test cases in each group of the two categories, and then the average execution time and the average fitness function value were computed for each group. The hill climbing approach was executed until it found the best solution that can be reached by the approach (i.e. a solution that cannot be of further improvement). On the other hand, our tabu search based approach ran for 50 iterations. A cut-off point had to be chosen because tabu search always moves to the point with the best fitness value of all the eight points around the square on each node even if the new point does not improve the current value of the fitness function and hence it would not run out of solutions.

Fig. 3 and Fig. 4 show bar charts of the results obtained from phase I. The charts clearly show the difference between the two methods in terms of the quality of the produced layouts and the execution time. The figures show that both methods give similar values for the fitness function with a slight advantage to our method. However, the execution time of our approach clearly outperforms the execution time of the hill climbing approach.

In phase II, we investigated the performance of approaches rather than the quality of the produced layouts. Therefore, the following process was performed to test which method has faster execution time when they reach similar values for the fitness function:

1. We ran the hill climbing method on the graphs until no improvements could be made on the value of the fitness function.
2. We ran our tabu search method until it reached an equal or better fitness function value compared to the one found by the hill climbing drawer.
3. We measured the execution time of the methods.

Fig. 5 shows bar charts of the results obtained from phase II. The columns obviously show that our drawer always finishes faster than the hill climbing drawer. These results indicate that excluding previously visited solutions from further investigation for a specific period of time is clearly an effective property in tabu search.

Finally, in phase III, we investigated the quality of the layout produced by the drawers rather than the performance. The following process was performed to test which method produces graph layouts with smaller values of fitness function when both drawers execute for the same period of time:

1. We ran the tabu search method on the graphs for 50 iterations. The execution time is computed and saved.
2. We ran the hill climbing method for the same period of time spent by the tabu search method.
3. We measured the value of the fitness function produced by the drawers in each of the above steps.

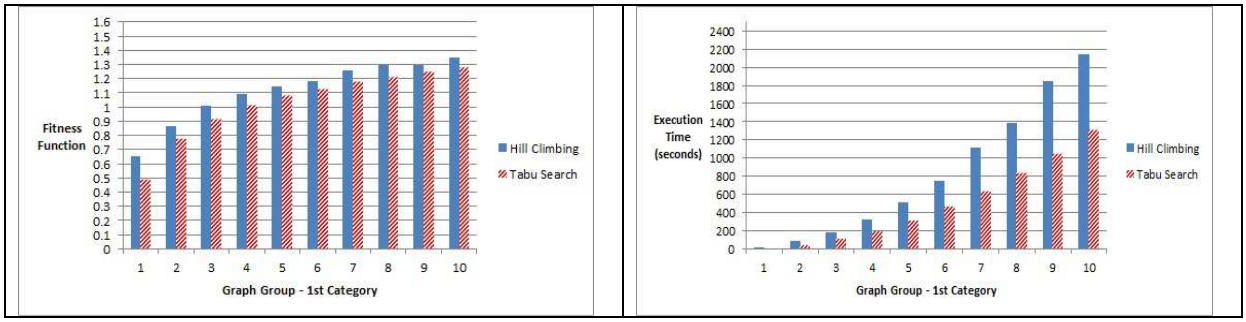

Fig. 3. Bar charts of the fitness function and execution time (in seconds) obtained by both methods when applied on the graphs of 1st category (Phase I)

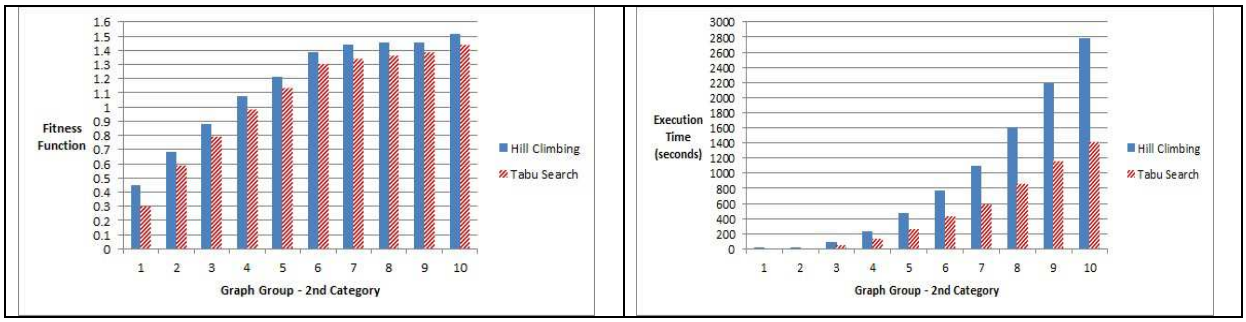

Fig. 4. Bar charts of the fitness function and execution time (in seconds) obtained by both methods when applied on the graphs of 2nd category (Phase I)

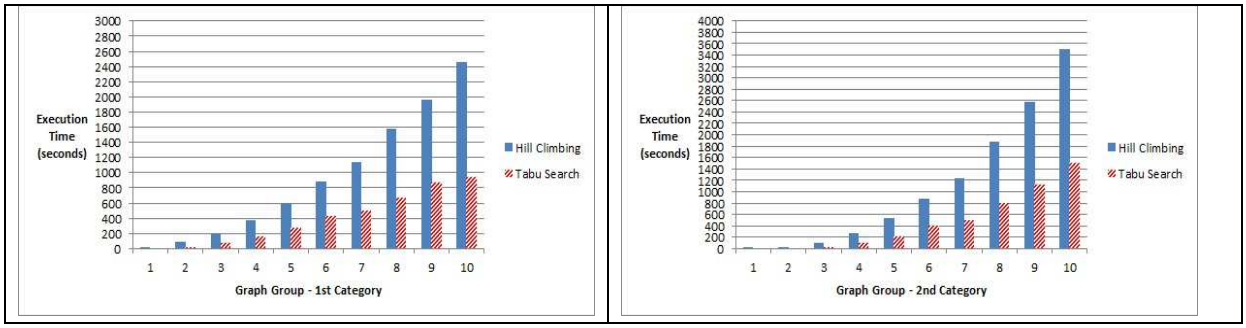

Fig. 5. Bar charts of the average execution time (in seconds) when the methods are applied on the graphs of the 1st and the 2nd categories (Phase II)

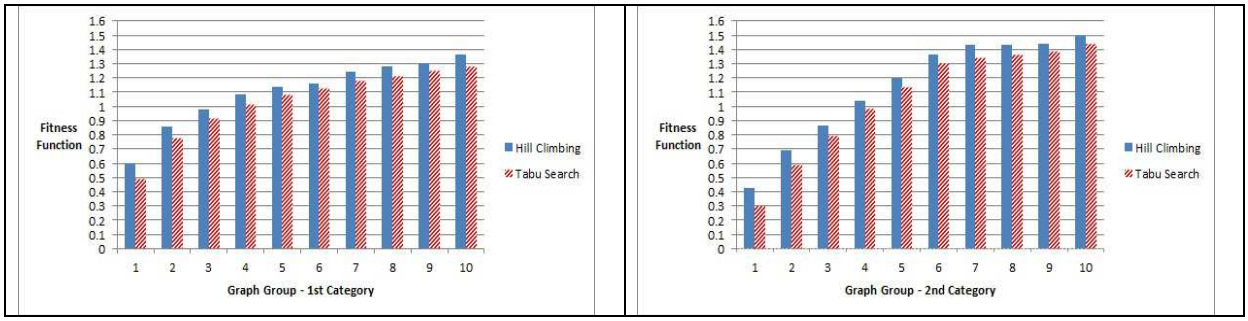

Fig. 6. Bar charts of the average values of the fitness function when the methods are applied on the graphs of the 1st and the 2nd categories (Phase III)

Fig. 6 shows bar charts of the results obtained from phase III. The columns look similar with a slight advantage to our tabu search based drawer. Therefore, we can conclude that our approach produces better graph layouts compared to hill climbing or similar layouts in the worst case when both drawers run for the same period of time.

Fig. 7 shows three different examples of graphs drawn by hill climbing approach and our approach.

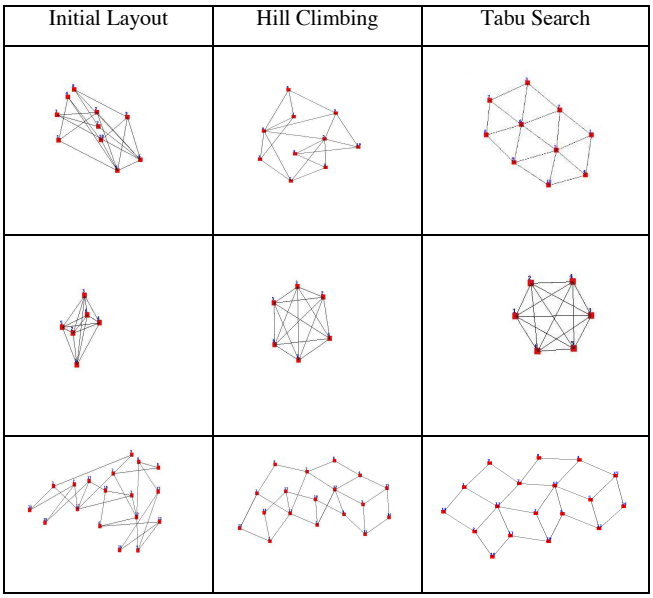

Fig. 7. Examples of graphs drawn by hill climbing and tabu search approaches

In terms of threats to validity, the algorithms used were deterministic and both used the same starting layout. The main internal threat seems to be in the implementation of the algorithms. Both methods were implemented by the same coder, and were run on the same machine. There is the possibility that one of the hill climber or tabu search was implemented in a more efficient way, however, the methods are sufficiently similar, sharing key code, so permitting some confidence that neither was particularly disadvantaged. In terms of external threats – that is to the generalizability of the results. We tested a number of randomly generated graphs that prevents selection bias (except in the parameters of the generation algorithm, such as number of nodes and edges). However, randomly generated graphs generally do not have the same characteristics as real world graphs, and so in the next section we explore the method applied to real world datasets sourced from the internet, although further real world testing is required to fully explore the generality of the results.

TABLE III
Real world graph datasets characteristics and sources

Graph	Nodes	Edges	Density	Source
1	34	78	0.139	[22]
2	62	159	0.084	[23]
3	105	441	0.081	[24]
4	112	425	0.068	[25]
5	115	613	0.094	[26]
6	198	2742	0.141	[27]
7	277	1918	0.050	[28]
8	297	2148	0.049	[29]
9	453	2025	0.020	[30]
10	500	13038	0.104	[31]
11	332	2126	0.039	[32]
12	415	7519	0.088	[33]
13	128	2075	0.255	[34]

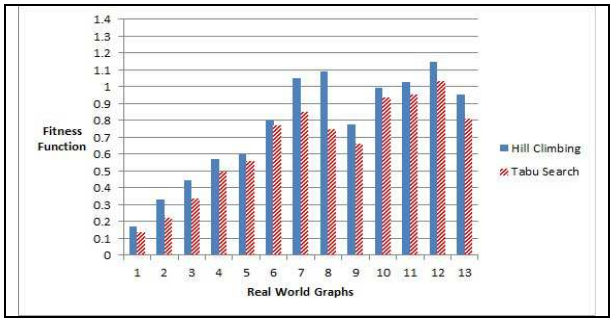

Fig. 8. Bar chart for the values of the fitness function of the two methods when applied on graph datasets in TABLE III

V. REAL WORLD GRAPH DATASETS

After performing several experiments on random graphs, we tested our system on real world graph datasets. We selected 13 different datasets from different sources as shown in TABLE III that also shows number of nodes, number of edges, and density in each graph. The graphs have different sizes with different densities. The initial layout of the nodes in each graph was generated randomly.

The results of the experiments are shown in Fig. 8 and TABLE IV. In the first two datasets only, the execution time of hill climbing is slightly faster than our tabu search based approach. This is due to the small size of those graphs. The results in the table demonstrate that our approach outperforms hill climbing approach in terms of execution time while the size of graphs increases. We also note from the figure that the values of the fitness function are always better in our approach regardless of the size of the graph.

Fig. 9 is an example of the layout produced by our drawer when applied on the first graph dataset in the list of real world datasets described in TABLE III.

TABLE IV
Execution time (in seconds) for both methods on graph datasets in TABLE III

Graph	Execution Time (seconds)	
	Hill Climbing	Tabu Search
1	0.699	0.931
2	1.405	1.826
3	11.822	9.421
4	16.234	9.671
5	18.192	15.543
6	468.838	298.526
7	258.042	149.060
8	323.139	185.277
9	347.227	193.338
10	14107.292	5968.619
11	384.990	210.799
12	4458.639	2190.103
13	257.392	178.600

VI. CONCLUSIONS AND FUTURE WORK

We have described an automated tabu search based approach for drawing general simple graphs with straight lines based on multi-criteria optimization. The method searches for

the best positions for the nodes that minimizes the value of the fitness function and draws a nice graph layout accordingly. Forbidding reverse moves, and the ability to escape local optima are two features that make tabu search a more effective layout method than hill climbing. Our experimental results on random graphs and real world graphs show that tabu search approach is faster than hill climbing, and in some cases the time is almost half, regardless of the size of the graph in terms of number of nodes and edges. On the other hand, both approaches produce layouts with good quality and in most cases tabu search approach slightly outperforms hill climbing.

In terms of future work, the definition of a tabu move might change. Instead of using absolute node position to determine whether a move is tabu or not, we might use its relative position. For instance, when a node moves to the left direction, its adjacent nodes should not move in the same direction, because moving them to the same direction is like shifting the whole sub-graph in one direction. Also, a graph clustering method can be used to divide the graph into sub-graphs where the nodes in each sub-graph would move according to their relative positions in their own cluster and each sub-graph would move according to its absolute position.

Also, it may be possible to develop a systematic way for choosing the values of the parameters used by our method. Several tests have been made with different values for `tabuDuration` and `tabuCutOff` parameters to come up with values that speed up the performance and produce nice graph layouts at the same time. More tests on different sets of graphs with different characteristics might lead to a clear process for choosing the values of the parameters.

Finally, the performance of our method may be further improved by implementing a hybrid of tabu search and other search based methods such as scatter search and path relinking. An interesting aspect of scatter search is that the approach performs a deterministic search instead of a random one. Path relinking, on the other hand, has the advantage of using previously encountered good solutions to obtain diversification and intensification in the search.

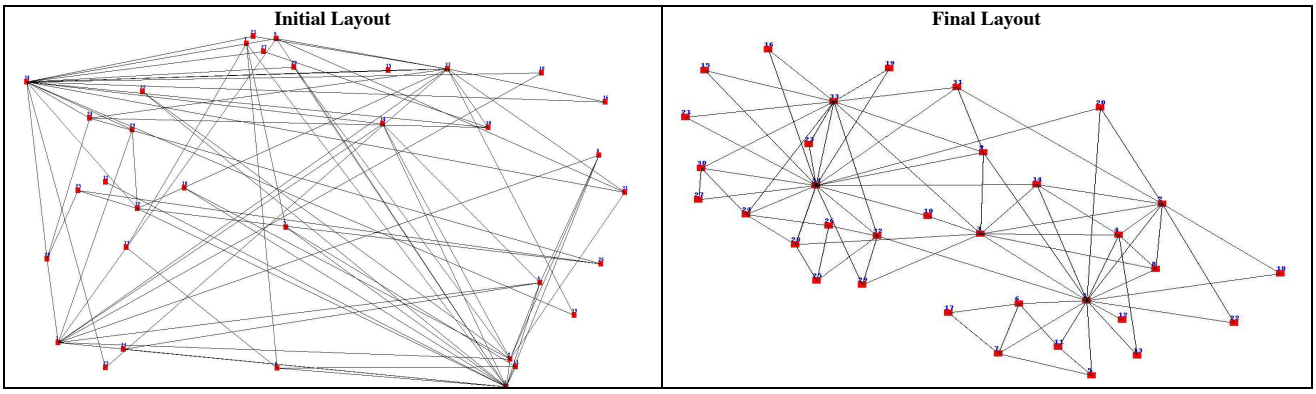

Fig. 9. Layout of graph dataset 1 (listed in TABLE III) produced by our method

REFERENCES

[1] H. C. Purchase, "Metrics for graph drawing aesthetics," Journal of Visual Languages and Computing, vol. 13, no. 5, pp. 501-516, 2002.

[2] R. Davidson and D. Harel, "Drawing graphs nicely using simulated annealing," ACM Transactions on Graphics, vol. 15, no. 4, pp. 301-331, 1996.

[3] J. Brank, "Drawing graphs using simulated annealing and gradient descent," Zbornik C 7. mednarodne multikonference Informacijska družba, pp. 67-70, 2004.

[4] C. Kosak, J. Marks and S. Shieber, "A parallel genetic algorithm for network diagram layout," San Diego, CA, USA, 1991.

[5] C. Kosak, J. Marks and S. Shieber, "Automating the layout of network diagrams with specified visual organization," IEEE Transactions on Systems, Man, and Cybernetics, vol. 24, no. 3, pp. 440-454, 1994.

[6] J. Branke, F. Bucher and H. Schmeck, "Using genetic algorithms for drawing undirected graphs," Proceedings of Third Nordic Workshop on Genetic Algorithms and their Applications, pp.193-206, 1996.

[7] T. Eloranta and M. Erkki, "TimGA: A genetic algorithm for drawing undirected graphs," Divulgaciones Matematicas, vol. 9, no. 2, pp. 155-171, 2001.

[8] A. Rosete-Suárez, A. Ochoa-Rodrıguez and M. Sebag, "Automatic graph drawing and stochastic hill climbing," In Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, pp. 1699-1706, 1999.

[9] J. Stott, P. Rodgers, J. C. Martinez-Ovando and S. G. Walker, "Automatic metro map layout using multicriteria optimization," IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 1, pp. 101-114, 2011.

[10] F. Glover, "Future paths for integer programming and links to artificial intelligence," Computer and Operations Research, vol. 13, no. 5, pp. 533-549, 1986.

[11] F. Glover and H. J. Greenberg, "New approaches for heuristic search: A bilateral linkage with artificial intelligence," European Journal of Operational Research, vol. 39, pp. 119-130, 1989.

[12] F. Glover, "Tabu search - part I," ORSA Journal on Computing, vol. 1, no. 3, pp. 190-206, 1989.

[13] A. Lim and Y. M. Chee, "Graph partitioning using tabu search," IEEE International Symposium on Circuits and Systems, vol. 2, pp. 1164-1167, 1991.

[14] A. Hertz, E. Taillard and D. De Werra, "A tutorial on tabu search," In Proceedings of Giornate di Lavoro AIRO, vol. 95, pp. 13-24, 1995.

[15] A. Hertz and D. De Werra, "Using tabu search techniques for graph coloring," Computing, vol. 39, no. 4, pp. 345-351, 1989.

[16] M. Laguna, R. Marti and V. Valls, "Arc crossing minimization in hierarchical digraphs with tabu search," Computers and Operations Research, vol. 24, no. 12, pp. 1175-1186, 1997.

[17] R. Marti, "A tabu search algorithm for the bipartite drawing problem,"

European Journal of Operational Research, vol. 106, no. 2, pp. 558-569, 1998.

[18] R. Marti and M. Laguna, "Heuristics and meta-heuristics for 2-layer straight line crossing minimization," Discrete Applied Mathematics, vol. 127, no. 3, pp. 665-678, 2003.

[19] A. BAYKASOGLU, S. OWEN and N. GINDY, "A taboo search based approach to find the pareto optimal set in multiple objective optimization," Engineering Optimization, vol. 31, no. 6, pp. 731-748, 1999.

[20] J. Pacheco and R. Martí, "Tabu search for a multi-objective routing problem," Journal of the Operational Research Society, vol. 57, no. 1, pp. 29-37, 2005.

[21] T. Thakur and J. Dhiman, "A tabu search algorithm for multi-objective purpose of feeder reconfiguration," Journal of Electrical and Electronics Engineering Research, vol. 3, no. 4, pp. 71-79, 2011.

[22] W. Zachary, "An information flow modelfor conflict and fission in small groups1," Journal of Anthropological Research, vol. 33, no. 4, pp. 452-473, 1977.

[23] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten and S. M. Dawson, "The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations," Behavioral Ecology and Sociobiology, vol. 54, no. 4, pp. 396-405, 2003.

[24] V. Krebs, "http://www.orgnet.com,", unpublished. Retrieved March 1, 2014.

[25] M. E. Newman, "Finding community structure in networks using the eigenvectors of matrices," Physical review E, vol. 74, no. 3, p. 036104, 2006.

[26] M. Girvan and M. E. Newman, "Community structure in social and biological networks," Proceedings of the National Academy of Sciences, vol. 99, no. 12, pp. 7821-7826, 2002.

[27] P. M. Gleiser and L. Danon, "Community structure in jazz," Advances in complex systems, vol. 6, no. 04, pp. 565-573, 2003.

[28] Y. Choe, B. H. McCormick and W. Koh, "Network connectivity analysis on the temporally augmented C. elegans web: A pilot study," In Soc Neurosci Abstr , vol. 30, no. 921.9, 2004.

[29] J. G. White, E. Southgate, J. N. Thomson and S. Brenner, "The structure of the nervous system of the nematode Caenorhabditis elegans," Philosophical Transactions of the Royal Society of London. B, Biological Sciences, vol. 314, no. 1165, pp. 1-340, 1986.

[30] J. Duch and A. Arenas, "Community detection in complex networks using extremal optimization," Physical review E, vol. 72, no. 2, p. 027104, 2005.

[31] J. Marcelino and M. Kaiser, "Critical paths in a metapopulation model of H1N1: Efficiently delaying influenza spreading through flight cancellation," PLoS currents, vol. 4, 2012.

[32] V. Batagelj and A. Mrvar, "Pajek datasets. Web page http://vlado. fmf. uni-lj. si/pub/networks/data," 2006.

[33] P. J. Taylor, World city network: a global urban analysis, Routledge, 2003.

[34] C. J. Melián and J. Bascompte, "Food web cohesion," Ecology, vol. 85, no. 2, pp. 352-358, 2004.

[35] M. G. Resende and C. C. Ribeiro, "GRASP: greedy randomized adaptive search procedures," in *Search Methodologies* , Springer US, pp. 287-312, 2014.

[36] R. Tamassia, Handbook of graph drawing and visualization, AMC, 10, 12, 2011.