Dynamic Parameter Spring Modeling Algorithm for Graph Drawing

Xuejun Liu and Buntarou Shizuki and Jiro Tanaka
Institute of Information Sciences and Electronics, University of Tsukuba,
1-1-1 Tennodai, Tsukuba-shi, Ibaraki-ken 305-8573, Japan
Email: {liuxj, shizuki, jiro}@is.tsukuba.ac.jp

ABSTRACT

This paper proposes a modified spring modeling algorithm for drawing undirected graphs, that is an extension of the forced-directed placement method of spring modeling algorithm by Eades. We call it ***dynamic parameter spring modeling algorithm.*** This algorithm is not only a general algorithm for graph layout, but also it is more suitable to get a ***semi-stable*** graph layout. A semi-stable layout is an intermediate layout, which leads to the final layout. The method of drawing this type of intermediate layout can shorten the cycle of the user's editing and speed up the process of layout in graph editing.

Keywords

Undirected graph, layout, graph drawing, drawing algorithm, graph editing.

INTRODUCTION

A graph is a simple, powerful, and elegant abstraction, which has a broad applicability with the visualization technique in computer science and many related fields. Force-Directed Placement is a well-known technique for drawing general undirected graphs [1-5]. Many models and algorithms have been proposed for the Force-Directed Placement, such as spring model by Eades[2], magnetic spring model by Sugiyama [6], and others[3-5]. These models and algorithms are just designed and applied for graph drawing in two-dimensional space. Lately, some algorithms have been extended to 3D space [7].

For visual presentations of undirected graph, the most important issue focuses on the speed of layout. Especially for a huge graph with a complicated structure with many nodes or vertices, the workload of computation becomes much larger and the algorithm looks heavier.

To speed up the process of layout for a given graph, in terms of different usage situations and different requirements from the user, various graph-drawing algorithms [8-10] have been proposed. Basically, these algorithms are aimed to get a final and stable graph layout.

On the other hand, we also found that the user is editing graph and only a little part of the graph is modified in most cases, such as adding or deleting one or more node(s), changing positions of a few nodes etc. In these cases, the final and stable graph layout is not needed since the user is just concerned with a graph layout without a high precision. We called this type of layout ***semi-stable*** graph layout.

In this paper, we propose a ***dynamic parameter spring modeling algorithm (DPSMA for short)*** to speed up the process of layout, which is based on the spring model and its algorithm. In DPSMA, we redefine the constant parameters in the spring model by dynamic parameters, and in the process of layout, these parameters are modified dynamically.

The remainder of this paper is organized as follows. Firstly, we describe the spring model and its problems for the undirected graph layout. Then in next two sections describe the ***dynamic parameter spring model*** and its algorithm and use it to speed up the general layout in detail and also semi-stable layout for graph editing. And then evaluations and examples are given. Finally, we give the concluding remarks.

SPRING MODEL AND ITS PROBLEMS

The spring modeling algorithm [2] for the spring model is a heuristic approach of graph drawing based on a physical system in which graph’s edges are replaced by springs and replace the vertices (nodes) by rings. Figure 1 shows the spring model. The forces acting on every node include spring force F_s and repulsion force F_r. The resultant of forces F_s and F_r can be calculated in terms of the formulas where d is the distance between a pair of nodes in the Figure 2. Under the influence of spring force between connected nodes and repulsion force between unconnected nodes, the graph will automatically layout until the system reaches a stable state.

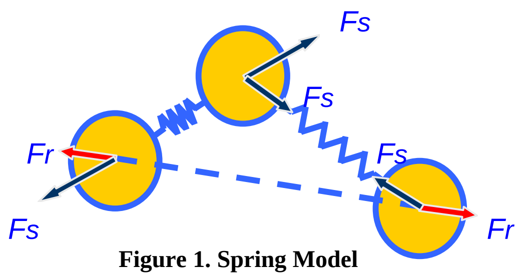

Figure 1. Spring Model

The basic idea of the spring modeling algorithm is to compute the resultant of forces, including spring forces and repulsion forces acting on every node, in iteration of the loop until the graph reaches a stable state. The time of a graph layout is determined by the numbers of iterations of the loop and the time of computation in iteration. If we can get a method where the times of iterations can be reduced without adding too much computation for iteration, the time of layout will decrease.

$$F_s = C_s \log(d / C_d)$$

$$F_r = C_r / d^2$$

Figure 2. Forces formula of Spring Mode

Figure 2 shows calculations of spring force and repulsion force in the spring model. The C_s , C_d and C_r are the constant parameters of the system. Among the parameters that control the forces acting on the nodes and causing their movements are spring length, spring stiffness, spring type and initial configuration. This very general heuristic can provide a solution for competing aesthetics.

When we use the spring modeling algorithm, there are two problems that we have to face.

1. Difficult to define the constant parameters C_s and C_r for the system.
2. Low running speed of the system.

Generally speaking, the layout of graph indicates the graph that has reached to a stable state according to the predefined terminal conditions by the user. In consideration of the speed of layout for a given graph is necessary to get a stable layout in any case, such as when the user is editing a graph. The aim of layout is just to help the user to observe, understand and operate the graph easily. A stable and final high-precision layout is only necessary after the edition or modification of the graph.

Sometimes the inability to specify the aesthetic criteria, used by the individuals, creates problems in understanding the graph. Nonetheless, for certain restricted classes of graphs, the expressions of aesthetic criteria of that graph can be specified. As for the semi-stable graph, as a intermediate, it is mainly used in editing or in cases where the user needs not to get the final stable graph. In general, according to the different layout criteria of final stable graph, the layout criterion of the semi-stable graph is also different. But for certain restricted classes of graphs, the aesthetic criteria of the semi-stable layout can also be specified for individual user or condition.

In the layout of a semi-stable graph, the main requirement of the user is just to get a rapid layout method to draw the semi-stable graph in which the aesthetic criteria can be basically guaranteed.

Therefore we should think about how to get a semi-stable graph layout quickly by spring modeling algorithm, and also for final stable graph. In most cases, the semi-stable layout is just for graph editing in which high precision is not needed and the most important thing is the speed of the layout.

To fulfill the above requirement, we modified the spring model to dynamic parameter spring model so as to speed up not only the semi-stable layout but also the final stable layout for a given undirected graph.

Based on the spring modeling algorithm, we propose a dynamic parameter spring model (*DPSM*) and define its algorithm called dynamic spring modeling algorithm to layout the undirected graph rapidly and also for the semi-stable layout.

DYNAMIC PARAMETER SPRING MODEL

Dynamic Parameter Spring Model and Force Definition

In spring model, in general, the bigger the parameters C_s and C_r are, the sooner the system reaches its stable state. But when C_s and C_r are too big, there will be a vibration phenomenon that will result the fact that the system can never reaches a stable state forever.

The parameters C_s and C_r of every node can influence the speed of the computation in every iteration. If we can find proper C_s and C_r for a given graph, the execution time of layout process will be reduced. Our idea is to give dynamic parameters C_s , C_r and adjust them for every node in the graph during the procedure of layout dynamically.

In our *DPSM*, we redefined the parameters of system C_s and C_r in the original spring model. In our definition every node in the system has its independent parameter C_s and C_r (Parameter C_d is still the common constant parameter.) (Figure 3.)

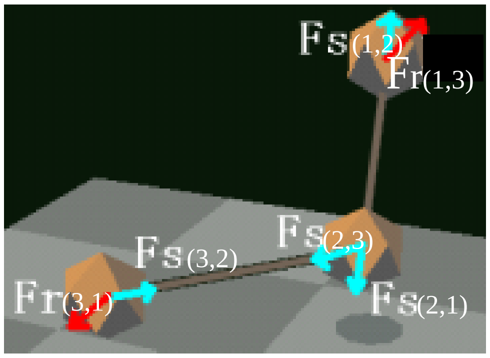

Figure 3. The Dynamic Parameter Spring Model

The new definition is given in Figure 4. The force acting on every node should be calculated by the parameters of the node itself. For any node i in the graph, the spring force acting on it by the spring between node i and node j is represented by $F_{s(i,j)}$, and the repulsion force acting on this node between node i and node j is represented by $F_{r(i,j)}$. C_{si} and C_{ri} have the same meaning with C_s and C_r in the original spring model respectively (Figure 1.). But C_{si} and C_{ri} are defined only for the node i and not for the whole system which is defined in the original spring model. The d is the same as in the original spring model.

$$F_{s(i,j)} = C_{si} \log(d / C_d)$$
$$F_{r(i,j)} = C_{ri} / d^2$$

Figure 4. Force formula for *DPSM*

Expected Position of Node

As far as layout of graph is concerned, after layout, every node in the graph will move to a stable position that is called final position. If we know the final position for every node or a position close to the final position, we can quickly move every node to that position. However, before layout, we do not know and cannot imagine where the final position for every node is.

According to the property of spring, after layout the distance between every a pair of nodes connected by a spring will be close to the length of spring. In a graph, if two nodes are not connected directly but a shortest path between two nodes can be found, we can imagine that after layout, the real distance between those two nodes must be close to the shortest path multiplied by the spring length.

In a real application, we want to draw a graph within the application window. To that purpose, we introduce a ***fixed-node.*** The fixed node is defined before layout and it ensure that at least some parts of the graph can be drawn insides the window. Both the user and the application designer can define the fixed node. We estimate there must be an ***expected position*** for every node and the distance between expected position and the fixed node is the shortest path multiplied by the length of spring.

The concept of shortest path is to find the shortest path in the graph from one node to another node. The algorithm will become different if the edges have different weights.

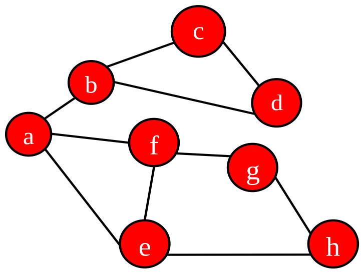

Figure 5. An example about shortest path

Figure 5 shows a small example of shortest path. In this graph, if the weights of all edges are the same, the shortest path between node a and node b is equal to 2, and the shortest path between node b and node h is 3.

In DPSMA, all pairs of shortest paths for nodes among the whole graph should to be computed before layout, and this computation is done only once.

Dynamic Parameters Definition

According to the discussion about expected position, we think that we can drive every node move quickly to the expected position, thus increase the speed of layout.

In terms of different position for a given node, we consider that two different groups of formula will be used to compute the dynamic parameters C_{si} and C_{ri} of every node.

If the real distance between fixed node and any node i is larger than the distance between fixed and expected position, for every node i, the C_{si} and C_{ri} will be

$$C_{si} = K_s * \frac{d_{f,i}}{C_d * SP_{f,i}}$$

$$C_{ri} = K_r * \frac{d_{f,i}}{C_d * SP_{f,i}}$$

respectively, where K_s and K_r are constant parameters predefined in initialization of program running. C_d is still the common constant parameter of system. $SP_{f,i}$ represents the shortest path between the fixed node and every node i. $d_{f,i}$ represents the distance between the fixed node and node i.

In another case, if the real distance between fixed node and any node i is smaller than the distance between fixed node and expected position of node i, the C_{si} and C_{ri} will be

$$C_{si} = K_s * \frac{C_d * SP_{f,i}}{d_{f,i}}$$

$$C_{ri} = K_r * \frac{C_d * SP_{f,i}}{d_{f,i}}$$

respectively. Therefore, we get the formulas for dynamic parameter computation in the Figure 6.

$$C_{si} = \max\left(\frac{d_{f,i}}{C_d * SP_{f,i}}, \frac{C_d * SP_{f,i}}{d_{f,i}}\right) * K_s$$

$$C_{ri} = \max\left(\frac{d_{f,i}}{C_d * SP_{f,i}}, \frac{C_d * SP_{f,i}}{d_{f,i}}\right) * K_r$$

Figure 6. Formulas for dynamic parameters

Figure 7 shows a simple example and we can use it to explain the formula defined in detail. The dotted circles represent the expected stable position of node i and node j. The node i is far away from the fixed node than the expected position of i. So the parameter will be $C_{si} = K_r * d_{f,i} / (C_d * SP_{f,i})$. For node j, it is close to the fixed node the its expected position, so we use the similar but different formula. $C_{sj} = K_s * (C_d * SP_{f,j}) / d_{f,j}$.

For these two situations, we use the maximum of the two ratios to adjust those parameters and speed up the computation.

The above describe the *DPSM*, and then we will discuss how to realize the rapid *DPSMA* in the next section.

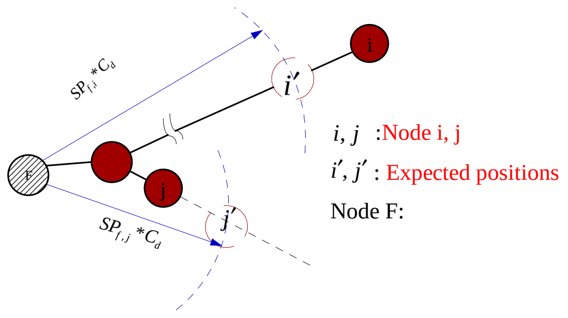

Figure 7. An example of *DPSM* graph

DYNAMIC PARAMETER SPRING MODELING ALGORITHM

Based on the definition of DPSM and formulas, the forces F_s, F_r are computed for all the nodes in the systems. After the computation, the movement of nodes will be made and the coordinates in x-axis, y-axis and z-axis will be modified. Finally the forces acting on every node will be computed until the system reaches a stable state.

In spring modeling algorithm, if the parameter C_s, C_r are too small or too big, some nodes will vibrate around their equilibrium position. In dynamic parameter spring model, we have adjusted the parameters C_s and C_r dynamically. As a result, when every node reaches its equilibrium position, the dynamic parameter will be of reasonable value, so the vibration phenomena can be avoided.

As for the *DPSMA*, to compute all the pairs of shortest path $SP_{f,i}$ between fixed node and any node i in a graph, we use the Floyd-Wars hall algorithm [11]. The input to the algorithm is the distance d_{ij}, which represents the distance between the node, i and node j (in the following code, $D[i][j]$) that is the shortest path between all pairs of nodes. The following lines of code represent the algorithm.

All pairs shortest path:

```
  Input adjacency matrix of graph with n nodes, W[0..
n-1][0..n-1].

  Initialize shortest path matrix, d[0..n][0..n];

  for (i=0; i<n; i++)

      for (j=0; j<n; j++)

          d[i][j]=W[i][j];

    for (k=0;k<n; k++)

          for (i=0; i<n; i++)
```

```
            for (j=0; j<n; j++)
                d[i][j] =min (d[i][j], d[i][k]+d[k][j]);
```

Dynamic parameter computing algorithm:

```
Compute $SP_{f,i}$;
Initialize const parameters $K_s$ and $K_r$;
for (i=0; i<n; i++)
{   compute $d_{f,i}$;
    if ($d_{f,i} < SP_{f,i}$){
        $C_{si} = K_s * d_{f,i} /(C_d * SP_{f,i})$;
        $C_{ri} = K_r * d_{f,i} /(C_d * SP_{f,i})$;
    }
    else{
        $C_{si} = K_s * (C_d * SP_{f,i}) / d_{f,j}$;
        $C_{ri} = K_r * (C_d * SP_{f,i}) / d_{f,j}$;
    }
}
```

In our *DPSMA*, the terminal condition of layout has also been defined. We defined a constant as the *force threshold* for a given graph in the beginning. When the biggest resultant forces among all nodes in the graph become smaller than the predefined force threshold, it means that the layout has reached to its stable state and the computation should be stopped. This is the terminal condition of the computations for a given graph.

For the semi-stable graph, the force threshold is needed to be defined a bigger constant that is decided by user in terms of user real requirements. In general, by experiments, the force threshold can be determined when the layout becomes acceptable.

EVALUATION AND EXAMPLE

We made some experiments to evaluate the performance of the *DPSM* and its algorithm.

5 undirected graphs have been used in three-dimensional space to make the evaluation. We defined the force threshold needed and the force threshold can determine the terminal condition.

In this experiment, the force threshold is set as 0.01(Newton), 0.05,0.1 and 0.5 respectively to evaluate the performance of DPSMA under different force threshold condition. The parameters K_s and K_r are set to be the same with the parameters C_s and C_r in the spring modeling algorithm respectively. The basic data of the 5 graph files are shown in Table 1.

Graph	1	2	3	4	5
Nodes	3	8	27	64	125
Edges	3	12	54	142	294

Table 1. Data of graphs in experiments

In our experiments, we compare the performance of the DPSMA with SMA in recursions times of layout, which is the number of recursion before the whole graph reach stability. We also compare the performance in real running time

No.	Force threshold	0.5	0.1	0.05	0.01
1	SMA	41	45	47	52
	DPSMA	29	34	36	41
	DPSMA/SMA	71%	75%	76%	79%
2	SMA	45	64	72	93
	DPSMA	34	51	53	74
	DPSMA/SMA	75%	79%	74%	79%
3	SMA	127	176	199	261
	DPSMA	89	149	169	222
	DPSMA/SMA	71%	84%	85%	85%
4	SMA	247	336	384	507
	DPSMA	176	264	311	386
	DPSMA/SMA	71%	78%	80%	76%
5	SMA	278	479	575	801
	DPSMA	193	367	432	637
	DPSMA/SMA	69%	76%	75%	79%

Table 2. Results of experiments

(Unit: recursion times)

From the Table 2, it is obvious that the iteration times in DPSMA are fewer then those in the original spring modeling algorithm. When the force threshold becomes bigger, the DPSMA will become more effective.

We also compare the real running time between SMA and DPSMA. Table 3 shows the results of real running time in mil-seconds.

No	Force threshold	0.5	0.1	0.05	0.01
1	SMA	151	160	181	192
	DPSMA	93	110	160	172
	DPSMA/SMA	62%	69%	88%	90%
2	SMA	230	280	350	472
	DPSMA	210	243	308	410
	DPSMA/SMA	91%	87%	88%	87%
3	SMA	171	232	297	431
	DPSMA	150	230	269	391
	DPSMA/SMA	88%	99%	91%	91%
4	SMA	4110	5740	6320	7981
	DPSMA	3696	5109	5610	7591
	DPSMA/SMA	90%	89%	89%	95%
5	SMA	49920	79833	92471	123869
	DPSMA	31978	56781	67684	93252
	DPSMA/SMA	64%	71%	73%	75%

Table 3. Results of experiments for 5 graph files

(Unit: mil-seconds)

From Table 3, we can get a similar conclusion in real running time with that derived from Table 2.

From Table 2 and Table 3 we also find that the increasing speed of spring modeling algorithm is faster than that of *DPSMA* with the increasing of nodes and edges between nodes. In other words, the *DPSMA* is more effective for a huge graph than the spring modeling algorithm.

When only the semi-stable layout of a given graph is required, in other words, the precision is not high, the bigger force threshold is, the more effect the DPSMA is than the spring modeling algorithm. From our experiences in experiments, when the force threshold is set to 0.5, the layout of graphs is acceptable. In general, we set to 0.5 the force threshold of our semi-stable layout in our experiments.

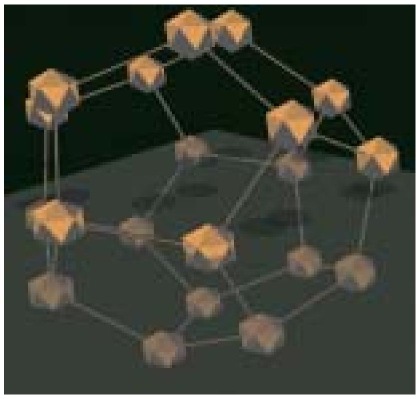

Figure 8. An example of undirected graph

Figure 8 is an example of undirected graph after automatic layout by our DPSM.

CONCLUDING REMARKS

We have proposed a modified spring model which we call *dynamic spring model*. It can be widely used in making a layout of undirected graph. Our idea is quite simple and intuitive. Based on the spring model and its algorithm, redefine the constant parameters in the spring model by dynamic parameters, and in the process of layout, these parameters are modified dynamically. This algorithm can speed up the procedure of layout and especially it is suitable to get a layout without high precision, such as for graph editing by user.

REFERENCES

[1] N. Quinn and M. Breur. A Force Directed Component Placement Procedure for Printed Circuit Boards. *IEEE Transactions of Circuits and Systems*, 26(6): 377-388, 1979.

[2] P. Eades. A Heuristic for Graph Drawing. *Congressus Numerantium* 42, 149-160, 1984.

[3] T. Fruchterman and E. Reingold. Graph Drawing by Force-Directed Placement. *Software – Practice and Experience 21*, 1129-1164, 1991.

[4] T. Kamada. Visualizing Abstract Objects and Relations, *World Scientific*, 1989.

[5] T. Kamada and S. Kawai. An Algorithm for Drawing General Undirected Graph. *Information letters*, 31(1): 7-15, 1989.

[6] K. Sugiyama and K. Misue. Graph Drawing by Magnetic-Spring Model, Res. Rep. ISIS-RR-94-14E, Inst. *Social Information Science*, Fujitsu Labs Ltd., 1994.

[7] T. Miyashita and J. Tanaka. Integrating Three-dimensional Spring Model and Augmented Directed Manipulation technique. *Transactions of IPSJ*, 42(3): 565-576, 2001(in Japanese).

[8] A. Frick, A. Ludwig and H. Mehldau. A Fast Adaptive Layout Algorithm for Undirected Graphs. *Proceedings of the Symposium on Graph Drawing GD'95*, Springer-Verlag, 389-403, 1994.

[9] R. Davidson and D. Harel. Drawing Graphs Nicely Using Simulated Annealing, *ACM Transactions on Graphics*, 15(4): 301-331, 1996.

[10] J. Nagumo and J. Tanaka. Introducing Fisheye-view into Graph Drawing Algorithm. *Transactions of IEICE*, 82-D-II(6): 1042-1048, 1999 (in Japanese).

[11] T.H. Corman., C.E. Leiserson. and R.L. Rivest, *Introduction to Algorithm*, MIT Press, 558-562, 1990.