A Model for the Visualization of Knowledge Acquisition

M. Delest[1], J.-P. Domenger[1], and L.J. van Gastel[2]

[1] LaBRI-Université Bordeaux 1, 351 Cours de la Libération, 33405 Talence, France
{maylis,domenger}@labri.u-bordeaux.fr
[2] Amstel Instituut, Universiteit van Amsterdam, Faculteit NWI,
Kruislaan 404, 1098 SM Amsterdam, The Netherland
gastel@wins.uva.nl

1 Abstract

Based on graph theory for fluid models, graphs and graph operations are introduced to model the acquisition of knowledge. A materialisation in software of this model can be used for automated advice about the navigation through the knowledge.

2 Introduction

A textbook about a subject can be viewed as a way of structuring knowledge. The chapters yield a division in parts, and subdivisions in sections and subsections occur if necessary. Such a textbook can be modeled by a graph by taking the chapters, sections, and the other parts as nodes and the inclusions of one part in another as edges. In fact this graph is a *tree*, as each section can only appear in one chapter.

However, in many cases the structure is more complicated than that of a tree. For instance, it might be that a particular section should be mastered before some other section in different chapter. So the latter depends on the first, and we will encounter this dependency as a reference in the textbook. So we have in fact a second type of edges, and the structure is that of a *directed acyclic graph* (DAG). Indeed, a cycle may not appear, as such a cycle in the knowledge structure would turn it impossible to acquire the knowledge.

The textbook gives more than just the DAG of organisation of the contents. The linear structure of a book, provides the reader with a path traversing the graph that respects the "depends on" relations. The numbering of the sections is usually done in such a way that the path is a depth-first traversal through the graph. Sometimes more paths are mentioned in the preface.

Thus we can split the representation of the knowledge structure of a textbook in a DAG and in a path. This is important, because in the process of knowledge acquisition, the path of a student will depend on several parameters, like the learning profile, the knowledge that the student already has, and the parts that the student wants to master. On the other side, the graph reflects the more

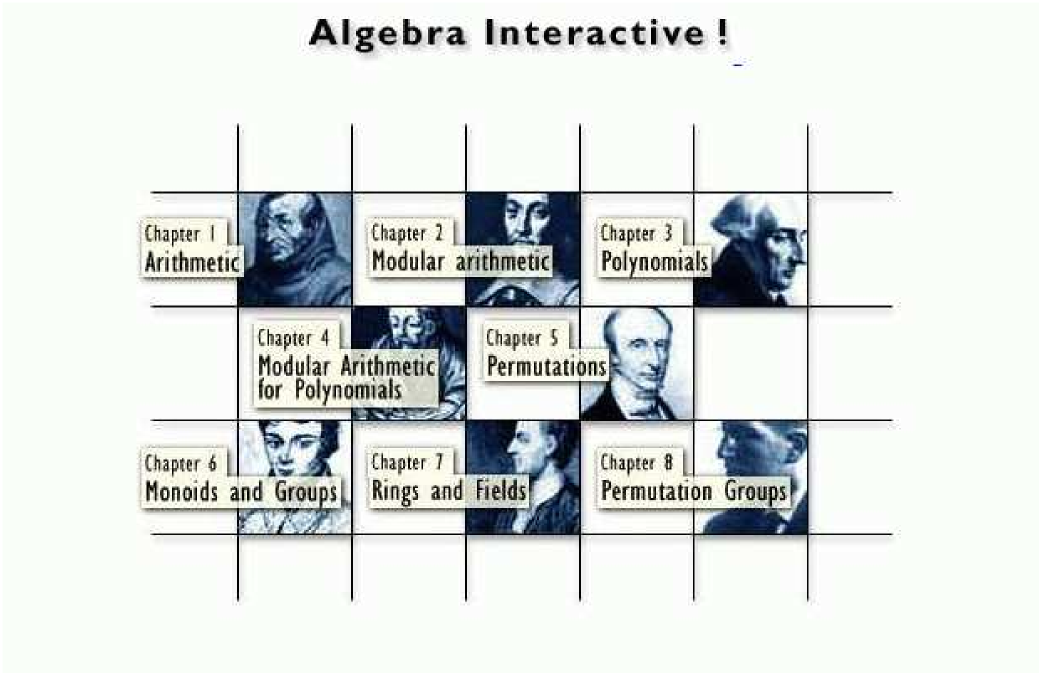

Fig. 1. Representation of the IDA Book.

intrinsic structure of the knowledge domain, as perceived by the domain experts. So the choice of the path is student-driven, while the design of the graph is domain-driven. Usually, the guide for the student is the content table. We show in figure 1, the content table of an interactive book on algebra [IDA]. This representation does not give any indication on the structure of the knowledge domain. In this paper, we choose a graph representation for whole book 9. Then, the importance of the nodes in the knowledge tree can be represented by a *valuation* on the graph: the leafs of the tree, which are the building blocks of the knowledge are valuated, and the other nodes obtain as valuation the total of the valuations of the leafs underneath. With exercises or tests it is often possible to measure how well a particular part is learned. In that case it is also possible to associate a valuation on the graph to the knowledge state of the student.

Moreover, by use of a particular redistribution of the valuations, the student can obtain information about the relevance of subgraphs where the internal dependencies and the current knowledge state of the student are taken into account. This information can be used as a *guide* for choosing the learning path: a personalized navigational advice. The algorithms to compute these valuations are a main result of this paper.

The overall aim of this paper is to define the mathematical objects and algorithms, to describe their properties, and to present a prototype implemented in Java.

3 The Graph Model

We associate to a book a tree T in the following way. The leaves of the tree L are the smallest units of knowledge, like theorems, exercises, etc. Chapters, sections, subsections are aggregates of these units and are therefore summits of subtrees. Let us denote these summits by S, so the set of all vertices of T are

$$V(T) = L \cup S.$$

The edges of T are the inclusion relations, so if a subsection v' is part of a section v, there is an edge (v, v'), see Figure 2. For each vertex v of T we denote by $S_T(v)$ the set of vertices that have v among its ancestors, so this is the subtree(s) of T starting at v except v itself.

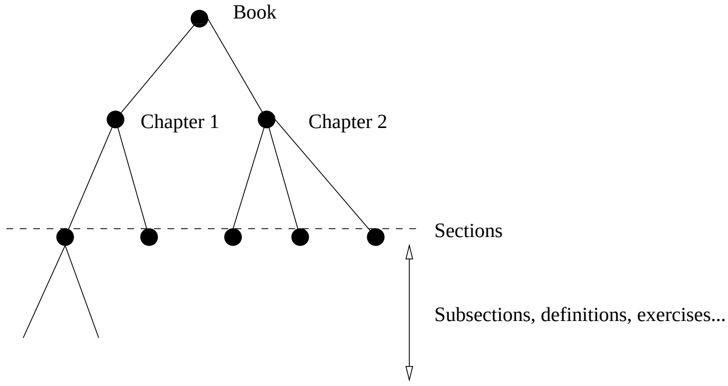

Fig. 2. The tree representation of a book.

Suppose $f: V(T) \mapsto \mathbb{R}^+$ is a map that represents the importance of the knowledge units. Such a map defines a valuation α_f on the graph by setting $\alpha_f(v) = f(v)$ for each leaf v, and for the summits:

$$\alpha_f(v) = f(v) + \sum_{v' \text{ child of } v} \alpha_f(v').$$

We will call a valuation *additive* with respect to f if the above formula holds. The value $\alpha_f(\text{root})$ represents the total importance of the knowledge.

A frequently occurring situation is that of a book with all elementary parts containing the knowledge. So the values of all leaves are set to a positive number and all other nodes to zero.

Let $g: V(T) \mapsto [0,1]$ be a map on the vertices of the tree that represents the fractions of the knowledge units that is mastered by a student. Let $\beta(v) = g(v) * f(v)$ for each vertex v in V(T). As before, we can associate an additive valuation α_β to the map β. The value $\alpha_\beta(\text{root})$ represents the total value of the acquired knowledge, and can be seen as a *mark*. It shows how well the student has mastered the subject in its totality.

Example 1. In Figure 3 any vertex v is labeled by $(f(v), \alpha_f(v), g(v), \alpha_\beta(v))$. All leaves have a knowledge value set to one. Other parts are set to 0. We suppose that the student already has mastered half of the part represented by the leftmost leaf l. Thus all the g values are 0 except $g(l)$ which is $1/2$. The result of the computation are displayed on each vertex.

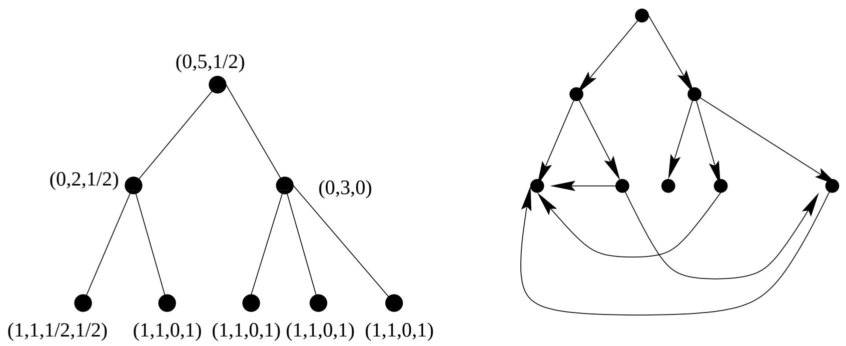

Fig. 3. An example of f, α_f, g, α_β valuations and of a DAG.

The difference of these valuations $\gamma = \alpha_f - \alpha_\beta$ is of interest for the student as it represents the knowledge that remains to acquire. This valuation γ is positive because f and g are positive and $g \leq 1$. Moreover, since α_f and α_β are additive, γ is additive with respect to the function $h = f - \beta$ as well.

If we take into account dependency relations in the knowledge, then the tree T becomes a *directed acyclic graph* D. The fact that a particular knowledge unit v' depends on a unit v will be represented by an oriented edge (v, v') from v to v'. In this way dependency relations are defined between leaves, but they make sense for the other vertices as well, as a summit of a subgraph stands for the aggregate of the knowledge of the leaves. The edges of this type are called *stepparent-stepchild* relations, so if (v, v') is an edge, v' is a stepchild of the stepparent v. We will write $\mathcal{E}_D$ for the set of these edges. So the graph D has the same vertices as T, but is has more edges (see Figure 3). If the tree is given a bottom-up orientation, then the direction of the arrows through the DAG is analogous to the flow of a river system. In particular the interpretation of an edge (v, v') is that the knowledge flow from v passes by the node v'. Acquiring knowledge is like rowing a boat upstream.

However, the algorithms are easier to describe when the tree is oriented top-down, beginning at the root and ending at the leaves. We will use this orientation in the rest of the article for the edges $\mathcal{E}_T$ of the tree.

So the graph D consists of the vertices of T and of the edges $\mathcal{E} = \mathcal{E}_D \cup \mathcal{E}_T$. See Figure 3 for an example of a DAG D.

With this orientation, the directed graph D may contain no cycles, as cycles in the knowledge structure graph would imply that you have to learn, say, v_1 before v_2, and v_2 before v_1. So indeed D is a directed *acyclic* graph. Moreover, the DAG verifies the following property

Property 1 *Let s, s' in $V(T)$ such that $s \notin S_T(s')$ and $(s, s') \in \mathcal{E}_D$ then*

$$\forall (u, v) \in S_T(s) \times S_T(s'), (v, u) \notin \mathcal{E}.$$

This states that if a section s' depends on a section s, then a reverse dependency of a subsection of s on a subsection of s' cannot exist. This property follows from the inclusion relations between sections and subsections.

4 Redirection of Flow

There is a way of manipulating the valuation γ, which measures the knowledge to acquire, to obtain a new valuation on the DAG that will help in navigating through the knowledge structure. We will describe this valuation in this section.

Consider the general situation of a DAG D that consists of a tree T and edges $\mathcal{E}_T$ and $\mathcal{E}_D$, and a map $h : V(T) \mapsto \mathbb{R}^+$ and an associated valuation α_h on T. Let σ be a map from $\mathcal{E}_D$ to $[0, 1]$ that satisfies for each vertex v:

$$\sum_{(v',v)\in\mathcal{E}_D} \sigma(v, v') \leq 1.$$

This map σ stands for the dependency weights. Because in a tree any vertex has only one parent, we can extend σ to all edges $\mathcal{E} = \mathcal{E}_T \cup \mathcal{E}_D$ by setting

$$\sum_{(v',v)\in\mathcal{E}} \sigma(v, v') = 1$$

for any vertex v.

We will use an iterative algorithm on D to define a map $f_0 \colon V(T) \mapsto \mathbb{R}^+$ that yields the new valuation α_{f_0}. Start by setting $f_0(v) = h(v)$ for each $v \in T$. Next we visit all the nodes of D with a depth first traversal of the directed graph D and change f_0 at each step.

- If a node has children or stepchildren that has not been visited yet, or if a node has no stepparents, go on with the traversal.
- Suppose we are at node v that has stepparents, then change f_0 with the following prescription. Let $v_1, ..., v_k$ be the stepparents, and let v' be the parent of v.

- set $f_0(v) = 0$,
- for the parent

$$f_0(v') = f_0(v') + f_0(v)\left(1 - \sum_{i=1}^{k} \sigma(v_i, v)\right),$$

- for the stepparents

$$f_0(v_i) = f_0(v_i) + f_0(v)\sigma(v_i, v),$$

Walking through the graph with the depth first traversal and this prescription yields f_0 and hence the valuation α_{f_0}. See Figure 4 for a single step. Let us roughly explain how to interpret this algorithm. If a vertex has stepparents then it can not be learned before its stepparents. Thus its f_0 value is set to 0. In this case its *knowledge* value is distributed to its parent and its stepparents. Let v_1, v_2 be two vertices of D such that $\alpha_{f_0}(v_1) > \alpha_{f_0}(v_2)$ then it is more effective to start to learn the vertex v_1 before v_2. This fact will be used for the navigational advice.

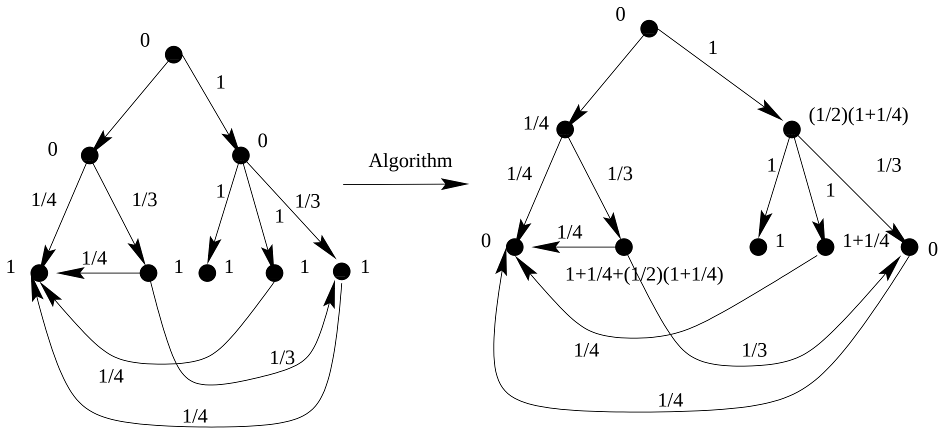

Fig. 4. A step in computing f_0.

The valuation α_{f_0} satisfies the following properties.

1. The value of α_{f_0} at the root of T equals that of α_h:

$$\alpha_{f0}(\text{root}) = \alpha_h(\text{root}).$$

2. At all nodes the Kirchoff law holds:

$$\sum_{v' \text{ child of } v} \sigma(v, v') * \alpha_{f_0}(v') = \sum_{v' \text{ parent of } v} \sigma(v', v) * \alpha_{f_0}(v).$$

The algorithm is displayed in Figure 5. It is clearly quasilinear in the number of vertices of D because of Property 1. Note that at each step, the contribution of v_i to the $f_0(v)$ valuation has been stored in the flow of the edge (v, v_i). In the following, we denote it by $(v, v_i).flow$

Now suppose that at some point a vertex is learned may be by doing exercices and getting a non zero score. The values of f_0 change when a node is learned. This change is described in the algorithm of figures **??** and has linear complexity. Let us briefly explain it.

Let v be the vertex that is learned to an extent of $l\%$.

First, we suppose that v is completely learned, i.e. $l = 1$. In such a case, the edges connecting v with its stepchildren must be removed using the algorithm `deleteEdges`. When an edge (v, v_i) that belongs to $\mathcal{E}_D$ is suppressed, then for all v_k parents or stepparents of v_i different from v, the map σ is updated according to:

$$\forall v_k \neq v, \sigma(v_k, v_i) = \sigma(v_k, v_i)/(1 - \sigma(v, v_i))$$

We show in Figure 6 what is happening when a node is completely learned and the algorithm in figure 7. Let $v_1, \ldots, v_n$ be the stepchildren of v. Then, when v is learned, the contribution of each v_i must be restored according to the learning percentage $l\%$. Thus, each v_i must be increased by a value equals to $(v, v_i).flow \times l\%$. Two cases may occur depending on the number of stepparent of vi after learning v. This treatment is performed recursively using the algorithm described in figure 8.

5 Visual representation of the DAG

These vizualisations were done with help of the program Latour[1] [HDM,CLY]. The goal of the java-application Latour is not to develop new layout algorithms but to concentrate on the issues raised by data exploration and interaction. The modular nature of Latour makes it easy to include new algorithms. It offers some classical tools as zoom, pan, fishe-eye, but also a visual cue for complexity. A so-called *metric* is calculated for each node of the tree. This metric represents the complexity of the sub-tree stemming at the node. Using the metric value and visuals tools like colour saturation and linewidth, Latour can highlight the weights of the edges. Different metric functions are possible and they may depend on the purpose at hand. Latour needs only a Java program that computes the metric. For the graphical representation of the tree T several methods can be considered, for instance:

- hierarchical representation [RT81],
- radial representation [E92],
- balloon representation [MH98].

When we tried these representations in Latour on the digital book IDA [IDA] (see Figure 9), it seemed that the balloon representation appeals most to readers and to authors. At the center is the node representing the book. On a first circle

[1] http://www.cwi.nl/InfoVisu

```
double ComputeChildrenFlow(Node v) {
    tmp = 0;
    for each v' child of v {
       if(edge(v,v').flow == 0) {
          double tmp1 = ComputeChildrenFlow(v');
          tmp += tmp1;
          v'.flow = tmp1;
        }
     v'.alpha    = 0.0;
     v'learnable = false;
    }
    return tmp;
}

void InitNodeEvaluation(Node v){
  if(v.mark == true) return;
  v.learnable = true;
  v.mark = true;
  v.f0 += v.f;
  v.g  = 0;
  for each v' stepchild of v
       NodeEvaluation(v');
  for each v' stepchild of v
       NodeEvaluation(v');
  if(v has stepparent){
       v.f0 = computeChildrenFlow(v)
       for each v' stepparent or parent of v {
           tmp   = sigma(v',v) * v.f0;
           edge(v',v).flow = tmp;
           v'.f0 += tmp;
       }
      v.f0 = 0;
  }
   else
     edge(v.parent,v).flow = 0;
  v.alpha = v.f0;
  for each v' child of v
    v.alpha += v'.alpha;
  v.beta  = v.alpha * v.g;
  if(v.learnable and v is leaf and v has no stepparent )
     v.learnable = true;
}
```

Fig. 5. The algorithm computing the initial valuation

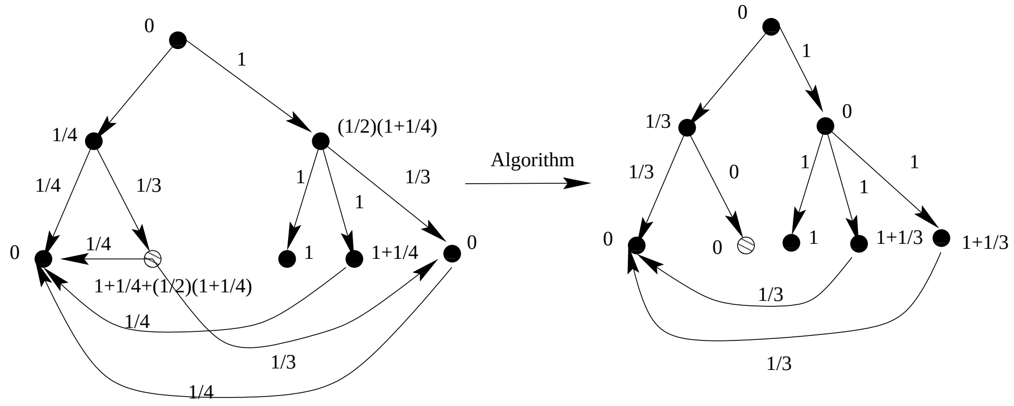

Fig. 6. The change on f_0 after learning the white vertex.

are the chapters. Around a vertex on the first level there is a smaller circle of vertex representing the sections of the chapter, and so on. One of the main interests of this representation is that the leaves of the tree are clearly separated. The nodes can be colored according to their state: red color for learnable nodes, blue color for not learnable nodes. The grey lines represent the stepedges.

The edges can be given a thickness according to the valuation of the child-vertex. If we do this for α_f, we get an impression of the importance of various parts of the book

For α_β we obtain the knowledge state of the student, and for γ we obtain the knowledge that remains to aquire. If we do this for α_{f_0}, then we get navigational advice: an idea of what is best to start studying with. Figure 9 shows this in the case that no nodes are learned yet. If you follow the thicker lines, then you start at the right topic that is here Chapter 1-"Arithmetics" and chapter 6-"Monoids and groups" . If these are learned, the lines will give you and advice what to do next. This is shown in the figure 10 where the learned topics that are choosen in chapter 6 appear in green in the figure.

The α_{f_0} valuation is implemented in Latour as such a *metric* using the algorithm given in Section 4.

6 Conclusion

We have implemented a version allowing interactions between the book server and the graph using servlet. In this version, when a node is selected on the graph, the corresponding page of the book is download in the navigator. Conversely, when a page is learned, the graph is updated. In future works, we plan to enlarge this prototype replacing nodes of the graph by icons and their size of the edges will be will depend on the graph valuation.

```
boolean computeVisibility(Node v){
   if v has stepparent
      return;
   else {for each v' children of v computeVisibility(v);}
   if v has no child
     v.learnable = true;
}
void NodeLearning(Node v, float learningPart) {
  knowledgeDistibuted = 0;
  v.g += learningPart;
  if(v.g == 1)
    for each v' stepchildren of v
      deleteEdge (edge(v,v'));
  for each v' stepchildren of v {
      tmp =  edge(v,v').flow;
      knowledgeDistibuted += tmp*learningPart;
      if(bottomUpF0Updating(v,v',tmp*learningPart))
        ComputeVisibility(v');}
   bottomF0Updating(v.father,v.f*learningPart);
   knowledgeDistributed  += v.f*learningPart;
    v.f0     -=  knowledgeDistributed;
    v.alpha  -=  knowledgeDistributed;
    v.beta    =  v.alpha * v.g;
}
void deleteEdge(edge(n,n')){
   supress n from  step parent of n'
   supress n' from  step children of n
   tmp = n.sigma(n,n')
   for each n'' stepfather of n'
     n''.sigma(n'',n') =  n''.sigma(n'',n') / (1 - tmp);
}
```

Fig. 7. The algorithm for one learning operation

References

[HDM] I. Herman, M. Delest, G. Melançon, Tree visualization and navigation clues for information vizualization, *Computer Graphics Forum*, 17 **2** 1998, 153-165.

[IDA] A.M. Cohen, H. Cuypers, H. Sterk, *Algebra Interactive!*, Springer-Verlag Berlin Heidelberg New York, 1999, ISBN 3-540-65368-6.

[CLY] I. Herman, G. Melanon, M.S. Marshall, D.J. Duke, M. Delest, J.-P. Domenger, Skeletal Images as Visual Cues in Graphs Visualization, IEEE/EG VisSym99 symposium, Vienna, mai 1999, eds. E. Grller, H. Lffelmann and W. Ribarsky, Springer-Verlag, pp. 13-22, 1999.

[RT81] E.M. Reingold, J.S. Tilford, Tidier drawing of trees, *IEEE Transaction on Software Enginiering*, 7 **2**, 1981, 223-228.

[E92] P. Eades, Drawing free trees, *Bulletin for Combinatorics and its Applications*, 5, 1992, 10-36.

```
boolean bottomUpF0Updating(Node learnedNode, Node v, double incValue){
 boolean learnable = true;
 if(v' has stepparrent)
     for each v' stepparent or parent v {
       if v' is not the learnedNode {
          edge(v',v).flow += edge(v',v).sigma * incValue;
          bottomUpF0Updating (learnedNode, parent, edge(v',v).sigma * incValue );
          v'.learnable = learnable = false;
     }
 else{
       if v.parent has stepparent {
          edge(v.parent, v).flow += incValue;
          v.f0 = 0.0;
          learnable = bottomUpF0Updating(learnedNode, parent, incValue);
       }
      else{
          v.parent.f0 -= edge(v.parent,v').flow;
          v.f0        += edge(v.parent,v').flow + incValue;
          v.alpha     += edge(v.parent,v').flow + incValue;
          v.beta       = v.alpha * v.g;
          learnable    = bottomUpF0Updating(v.parent, incValue);
       }
     }
 if v is the root {
      v.f0    += incValue;
      v.alpha += incValue;
      v.beta       = v.alpha * v.g;
  }
  return learnable;
}
```

Fig. 8. The bottom-up updating

[MH98] G. Melanon, I. Herman, Circular drawings of rooted trees, Technical Report INS-R9817, CWI Amsterdam, 1998.

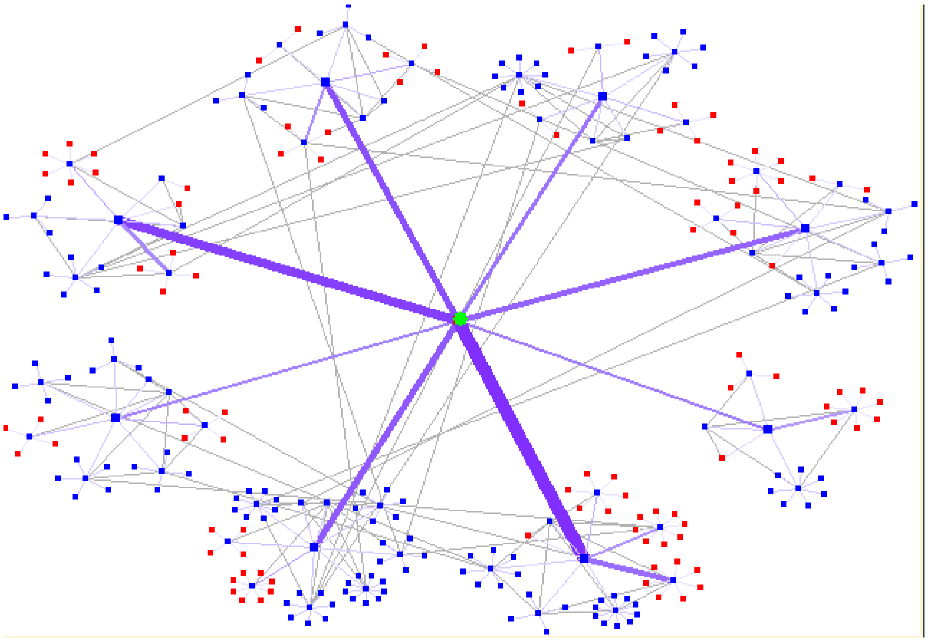

Fig. 9. Initial drawing for IDA graph.

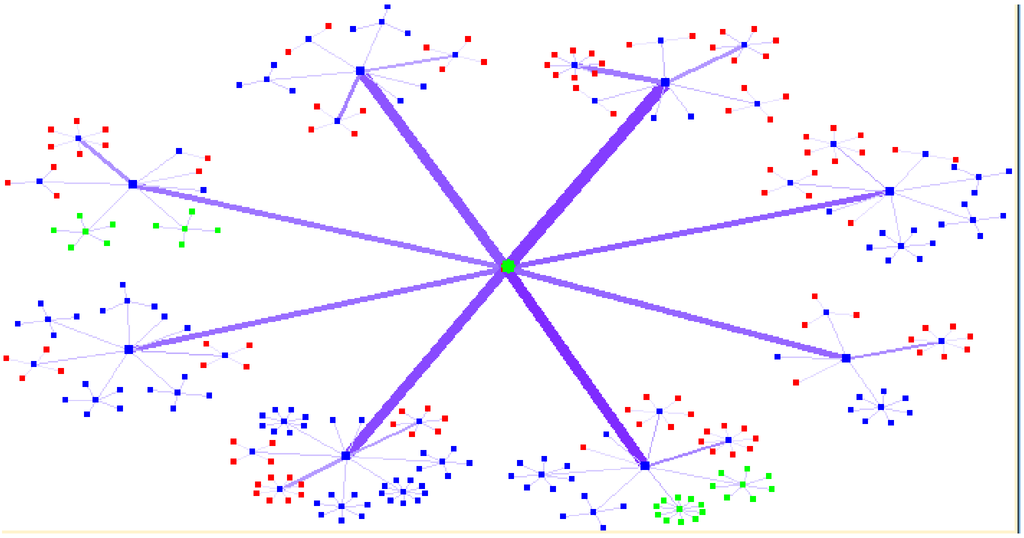

Fig. 10. The IDA graph after several learning operations.