Automatização do desenho de grafos e sua aplicação em Inteligência Artificial*

Paulo Quaresma† pq@fct.unl.pt
José G. P. Lopes gpl@fct.unl.pt

Centro de Inteligência Artificial, UNINOVA
Quinta da Torre
2825 Monte da Caparica
Portugal
27 de Julho de 1991

Resumo

A necessidade de visualizar estruturas de dados complexas cria a exigência da geração automática de novos objectos gráficos. Neste artigo será apresentado um novo objecto gráfico — *o Graph widget*[1] — que foi criado na hierarquia de classes do X-Windows. Este widget suporta a criação e a manipulação grafos recorrendo a uma das três funções de desenho implementadas: árvores, grafos hierárquicos e grafos gerais. No artigo são descritas as alterações feitas aos algoritmos originais sendo também descrita a interface realizada para o Prolog de modo a permitir a utilização do objecto em ambientes de programação em lógica. Finalmente, são discutidas as possíveis aplicações do widget e alguns dos problemas em aberto.

1 Introducção

Os grafos têm sido utilizados para modelar a realidade e para representar as relações entre os objectos em diversos campos da Informática. Nos últimos anos a necessidade de visualizar automaticamente os grafos criados tem aumentado consideravelmente com o desenvolvimento de interfaces homem-máquina e com a utilização de ambientes integrados de programação. Por outro lado, as representações gráficas em ambientes de programação permitem uma fácil compreensão dos objectos e diminuem o tempo de desenvolvimento e de depuração de erros.

No entanto, são necessários algoritmos apropriados para se produzir automaticamente desenhos de grafos esteticamente correctos. Em [ET89, TBB88] é

*Este trabalho foi suportado pela JNICT, através dos projectos "Comunicação em Português com Computadores", contracto n^o PMCT/MIC/87439 e "Compreensão Robusta do Português", contracto n^o PMCT/P/TIT/167/90; INIC através do projecto CALIPSO; FCT/UNL e Gabinete de Filosofia do Conhecimento

†É actualmente bolseiro da JNICT, referência n^o PMCT/BIC/152/90

[1]Widget ↔ Window Gadget, a abstracção do X-Windows para objectos gráficos

apresentado um resumo dos algoritmos desenvolvidos até à data e muitos dos problemas ainda existentes, sendo os algoritmos classificados de acordo com as suas caracteríticas. Na secção 2 é apresentada uma classificação ligeiramente diferente para os algoritmos de visualização de grafos (adaptada e modificada de [TBB88]) que foi utilizada para definir e caracterizar o trabalho realizado.

Para suportar as estruturas de dados do tipo grafo e, dado que é utilizada a linguagem Prolog e o X-Windows como ambiente de trabalho no Centro de Inteligência Artificial do UNINOVA, foi decidido criar um novo objecto gráfico na hierarquia de classes do X-Windows e construir uma interface para o Prolog. O novo objecto — *Graph widget* — foi definido como uma subclasse do X e implementado em C utilizando o toolkit do X [MAS88] (ver secção 3). Tentou-se construí-lo o mais geral possível de modo a permitir a sua futura utilização em diferentes campos da Informática. Foi ainda implementada uma interface para o X-Prolog, um ambiente Prolog que permite o acesso às funcionalidades do X-Windows (ver [Abr89]), e um editor de grafos em Prolog.

2 Classificação dos algoritmos de visualização de grafos

De modo a compreender completamente as decisões tomadas durante a implementação do widget é necessário, em primeiro lugar, caracterizar e classificar os algoritmos existentes.
Os algoritmos de visualização de grafos podem ser classificados em relação a:

1. classe dos grafos (árvores, hierárquicos, planares, dirigidos e não dirigidos)
2. standard gráfico (linhas rectas, grelha, polígonos, etc.)
3. critérios estéticos
4. restrições
5. complexidade computacional
6. características dinâmicas/estáticas

Os critérios estéticos definem os critérios de desenho utilizados pelos algoritmos e podem ser de diferentes tipos:

- área de desenho
- equilíbrio entre os eixos horizontal e vertical
- curvas dos arcos
- faces convexas
- cruzamentos entre os arcos
- comprimento dos arcos
- simetria

- densidade dos arcos no desenho
- verticalidade de estruturas hierárquicas

Embora se possa caracterizar o desenho de um grafo específico pela escolha dos critérios estéticos, não é possível deste modo suportar conhecimento semântico. Este tipo de conhecimento necessita da definição de restrições ao grafo, que podem ser de vários tipos:

- forma dos subgrafos
- dimensão dos nós
- localização de um dado conjunto de nós, etc.

3 Graph Widget

De acordo com a secção 2, tiveram de ser feitas várias decisões ácerca da funcionalidade do widget graph de modo a:

- suportar qualquer tipo de grafo,
- ser dinâmico de modo a permitir a criação e supressão de nós e arcos,
- permitir a visualização parcial do grafo,
- permitir a definição de restrições.

A classe do widget *GraphWidgetClass* foi criada como subclasse de *Constraint*, permitindo deste modo aos objectos terem filhos (i.e. nós) com restrições.

Os recursos do grafo e as restrições dos nós são descritas brevemente nas subsecções 3.1 e 3.2. Em [Qua90] é dada uma descrição completa do widget.

3.1 Recursos do grafo

Os recursos definidos permitem a obtenção das seguintes funcionalidades:

- escolher e/ou definir a função de desenho. Pode ser uma das três funções pré-implementadas (secção 3.3) ou pode ser uma função definida pelo utilizador.
- alternar entre o modo automático e manual. Em modo manual o utilizador é responsável pelo cálculo das posições dos nós.
- definir a distância por defeito entre os nós.
- definir se o grafo é dirigido ou não.
- definir o comportamento do grafo quando os nós alteram de tamanho.
- seleccionar um widget onde a totalidade do grafo será desenhada e que permitirá ao utilizador visualizar ao mesmo tempo o grafo parcialmente e na sua totalidade.

3.2 Restrições do grafo

Os nós do grafo podem ter restrições que permitem a definição das seguintes características:

- lista dos nós a que estão ligados.
- identificação do contexto gráfico associado a cada arco, permitindo a definição de diferentes estilos para cada arco.
- pista para a posição vertical e horiziontal do nó. Em modo manual este valor é usado como posição final.

3.3 Funções de visualização

Foram implementadas três funções de visualização de grafos:

- Árvores,
- Grafos hierárquicos,
- Grafos gerais.

Estas opções foram baseadas nas seguintes considerações:

- permitir a visualização automática de grafos gerais mas, também, de grafos específicos (árvores e grafos hierárquicos).
- não foi implementado um algoritmo para grafos planares devido à sua complexidade computacional. O trabalho de Chiba [CNAO85, CON85] requere, no entanto, uma consideração e atenção especial.

3.3.1 Árvores

O algoritmo de Reingold e Tilford ([RT81]) está definido para árvores binárias e satisfaz os seguintes critérios:

- cruzamento entre os arcos
- verticalidade
- simetria
- área
- isomorfismo entre as sub-árvores

e pode ser descrito assim (de [TBB88]):

se a árvore é nula ou é um só nó

então construir o seu desenho

senão recursivamente desenhar as sub-árvores esquerda e direita; colocar as duas sub-árvores juntas de modo a que a mínima distância entre elas seja δ; colocar a raiz entre as sub-árvores

fimse

Este algoritmo foi generalizado de modo a suportar árvores n-árias e até grafos gerais (no entanto desenhados como árvores). Por outro lado, como se desejava um algoritmo dinâmico, foi alterado de acordo com o trabalho de Moen [Moe90]. Permite, agora, nós de qualquer formato e a sua inserção e supressão. Ver figuras 1 e 2 com exemplos de grafos hierárquicos e gerais desenhados utilizando este algoritmo.

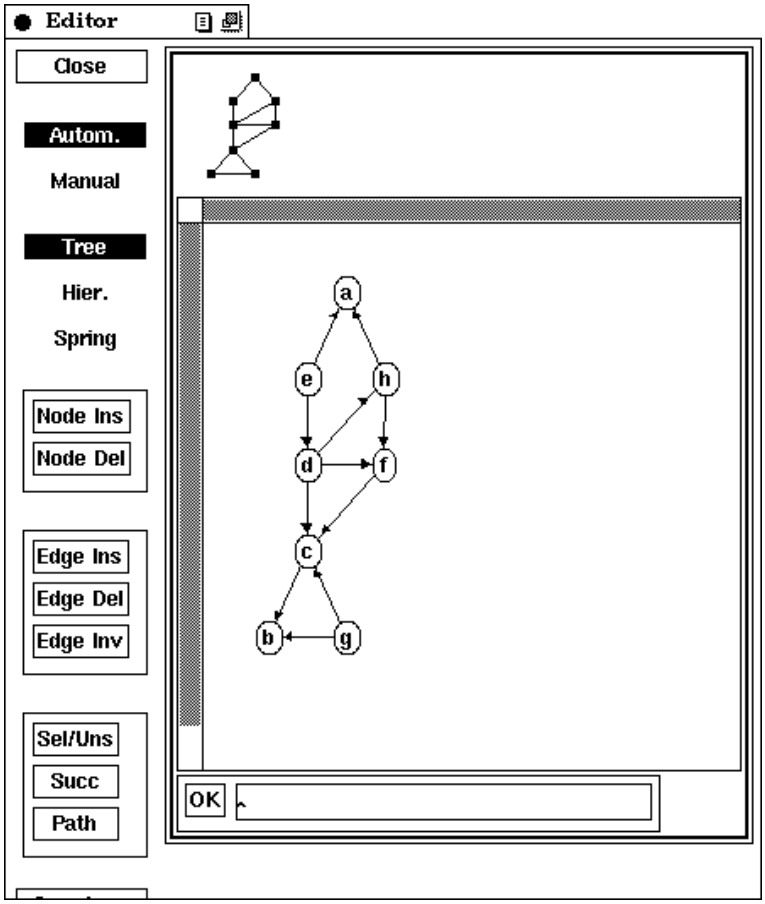

Figura 1: Grafo A desenhado pelo algoritmo de árvores

3.3.2 Grafos hierárquicos

Como algoritmo para representar grafos hierárquicos escolheu-se o algoritmo proposto por Sugiyama [STT81] de modo a satisfazer os seguintes critérios estéticos:

- cruzamento entre os arcos

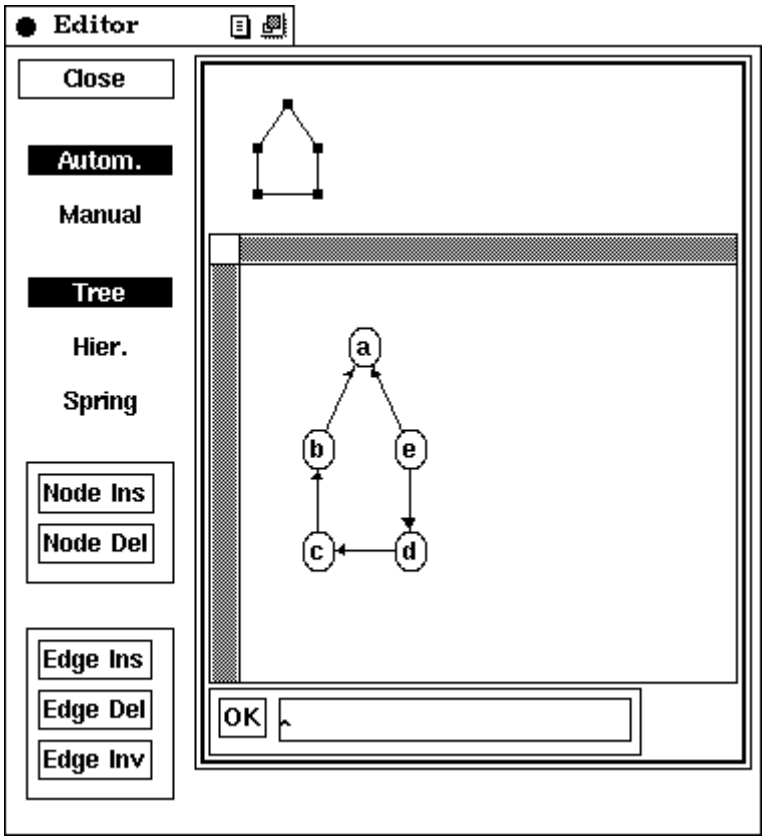

Figura 2: Grafo B desenhado pelo algoritmo de árvores

- verticalidade
- simetria
- comprimento dos arcos
- densidade dos nós

que pode ser resumido da seguinte forma:

1. atribuir níveis aos nós
2. ordenar os nós de cada nível de modo a minimizar o número de cruzamentos entre os arcos
3. posicionar os nós de modo a melhorar a visualização

Na primeira fase são atribuidos níveis aos nós de modo a que cada nó esteja num nível inferior aos seus antecessores. Como o algoritmo original não suportava ciclos, utilizou-se a sugestão de Rowe [RDM+87] para inverter temporariamante os arcos. Nesta fase os arcos longos (que ligam nós separados por mais de um nível) são divididos em vários arcos de um só nível através da criação de nós fantasma.

Na segunda fase o algoritmo tenta minimizar o número de cruzamentos de arcos fazendo múltiplas passagens ao longo do grafo e alterando a ordem dos nós. Sugiyama sugere a ordenação pelo baricentro superior e inferior de cada nó — posição média dos predecessores (sucessores) imediatos de um nó. Por outro lado, Gansner [GNV88] propõe um pré-processamento de cada nível.

Como heurística foi utilizada uma combinação do trabalho de Gansner e de Rowe. Cada nível é pré-processado tentando diminuir o número de cruzamentos. Depois, é utilizada uma modificação da heurística de Rowe — após a primeira passagem, os níveis são ordenados pela média pesada da posição do nó e do baricentro superior e inferior. Em geral esta técnica levou a resultados superiores, embora não tenha sido feito um estudo comparativo suficientemente detalhado.

A terceira fase determina a posição final de cada nó, tentando reduzir o número de curvas dos arcos e verticalizar o desenho final.

Este algoritmo suporta grafos gerais mas, em geral, tem resultados fracos para grafos cuja estructura não seja hierárquica.

As figuras 3 e 4 mostram os mesmos grafos desenhados na secção anterior mas utilizando desta vez o algoritmo hierárquico.

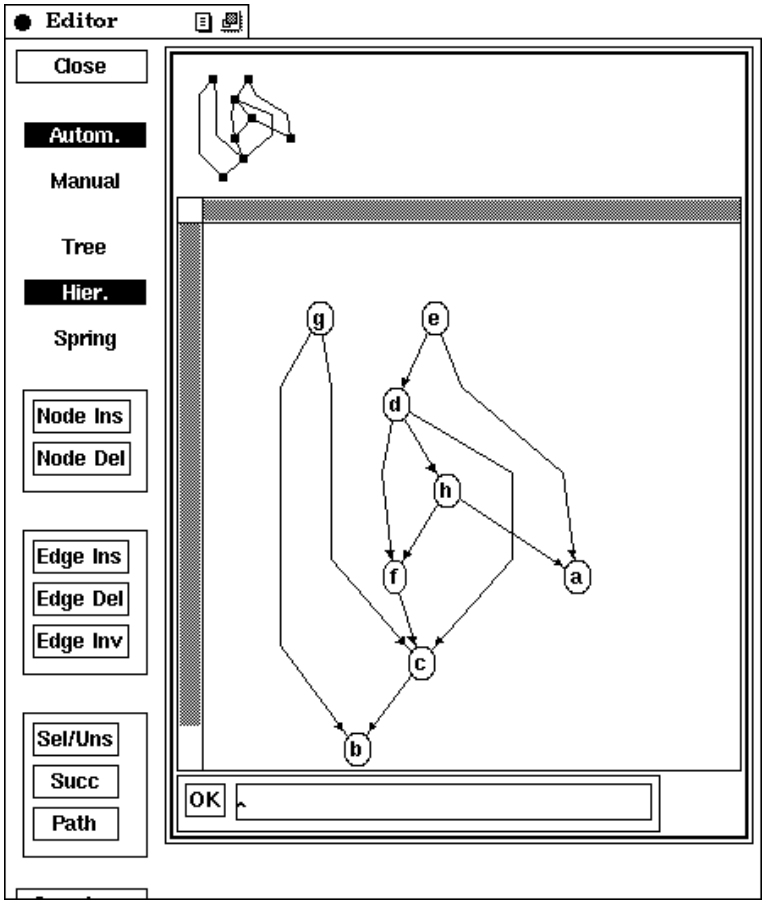

Figura 3: Grafo A desenhado pelo algoritmo hierárquico

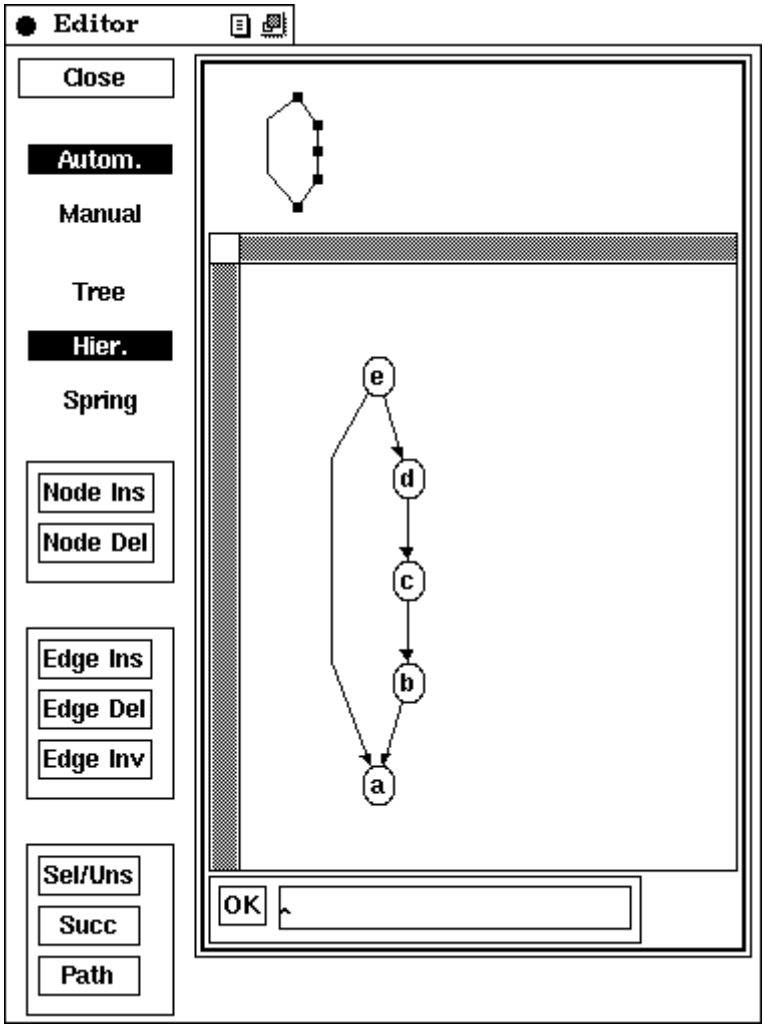

Figura 4: Grafo B desenhado pelo algoritmo hierárquico

3.3.3 Grafos gerais

Há vários algoritmos ([ET89]) para desenhar grafos gerais. Por um lado, Tamassia [TBB88] descreve um algoritmo determinístico e complexo com diversas fases (planarização, ortogonalização, compactação e desenho); por outro lado, Kamada [KK89] introduz um sistema virtual dinâmico em que cada par de nós está ligado por uma dada força e em que se tenta atingir um estado de energia mínima.
Escolheu-se implementar o algoritmo de Kamada devido a:

- ser dinâmico (suporta bem pequenas alterações à estrutura do grafo),
- satisfazer vários critérios estéticos (equilíbrio, densidade, simetria e isomorfismo),
- não ser demasiado complexo,
- poder usar informação dada pelo utilizador.

Para Kamada a energia total do sistema é dada pela seguinte equação:

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{ij} (\mid p_i - p_j \mid - l_{ij})^2 \qquad (1)$$

onde $p_1, \ldots, p_n$ são as partículas do plano correspondentes aos nós $n_1, \ldots, n_n$; l_{ij} é o comprimento desejado entre os dois nós que é calculado a partir do comprimento do caminho mais curto entre n_i and n_j; k_{ij} é a força existente entre n_i and n_j e é inversamente proporcional ao comprimento do caminho mais curto entre os nós.

O objectivo do algoritmo é minimizar a equação 1. Para tal Kamada propõe um sistema baseado no método de Newton-Raphson que calcula um mínimo local de $2n$ equações não lineares. O algoritmo é iteractivo e termina quando atinge uma posição estável.

Durante a fase de implementação foram detectados alguns problemas, nomeadamente a atribuição de valores às constantes utilizadas pelo algoritmo demonstrou ser um ponto crítico. Este problema foi resolvido através da utilização de um dos recursos descritos na subsecção 3.1; os valores podem ser diferentes de grafo para grafo e de desenho para desenho.

Também foi detectado que em alguns casos a iteraccão entrava em ciclo. Foi necessário construir um detector de ciclos mais potente: o algoritmo pára quando atinge um ponto estável ou quando entra em ciclo entre alguns pontos instáveis.

As figuras 5 e 6 mostram os mesmos grafos desenhados nas secções anteriores.

4 Interface para o Prolog

Uma das motivações deste trabalho foi contruir uma interface para a linguagem Prolog de modo a permitir utilizar este novo objecto em ambientes de programação em lógica. Existia a necessidade de ferramentas que permitissem a visualização e a manipulação de grafos (necessárias à construção de browsers, editores , ...).

Recorremos ao ambiente avançado para programação em lógica ALPES[2] usando o Quintus Prolog sobre o X-Windows (ver [Abr89]). Este ambiente tem definida uma linguagem de descrição de widgets (WDL) que permite especificar, criar e enviar mensagens aos objectos ([Abr89]).

Para implementar a interface apenas foi necessário extender a WDL de modo a suportar o novo widget e definir os tipos e os procedimentos exportados pelo widget. Em [Qua90] o código completo é listado.

Para demonstrar o resultado final foi implementado um pequeno editor de grafos com algumas funções básicas (criação e supressão de nós e arcos), sendo as figuras deste artigo desenhadas com este editor.

[2]Este ambiente foi desenvolvido no âmbito do projecto ESPRIT 973

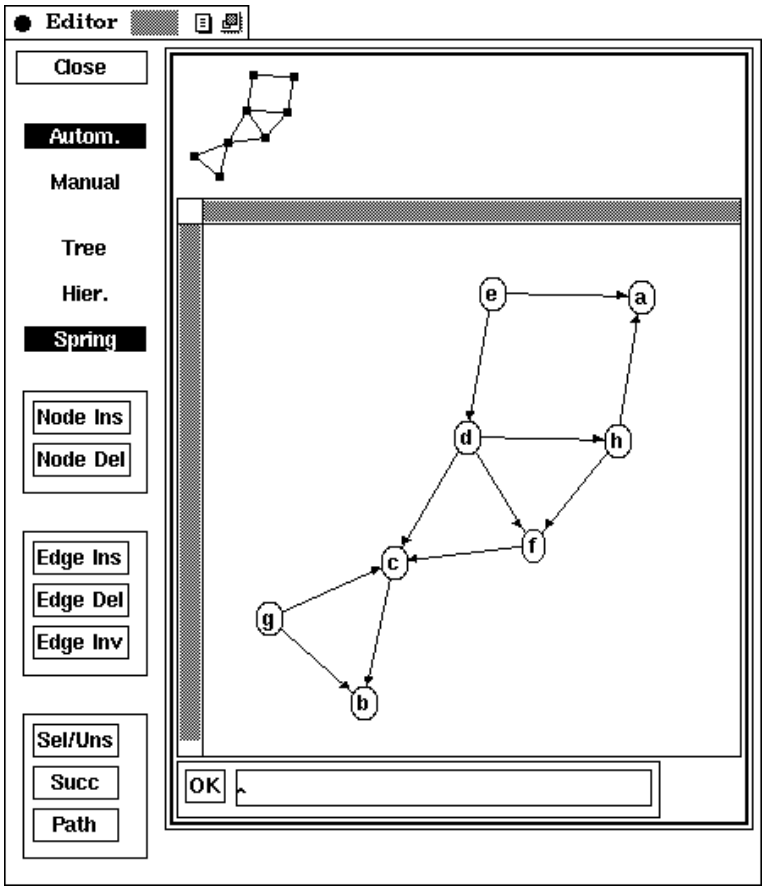

Figura 5: Grafo A desenhado pelo algoritmo de Kamada

5 Conclusões e trabalho futuro

O widget criado é uma ferramenta para a visualização e manipulação de estruturas de dados complexas com as seguintes características principais:

- é geral,
- permite uma configuração pelo utilizador,
- é utilizável em ambiente Prolog,

Estas caracerísticas permitem que seja usado como base para outras camadas de software. De facto, pode ser uma ferramenta base para o desenvolvimento de ambientes integrados de programação em lógica pois pode ser utilizado para implementar várias ferramentas de desenvolvimento e de depuração (browsers, grafos de execução de programas, editores de grafos, etc).

De facto já foi usado na construção de um editor de grafos conceptuais [Wer91] e no projecto ESPRIT CIM-PLATO[3] n^o 2202 como ferramenta de desenvolvimento de interfaces [PPM91].

[3]European Strategic Programme for Research and Development in Information Technology - Computer Integrated Manufactoring

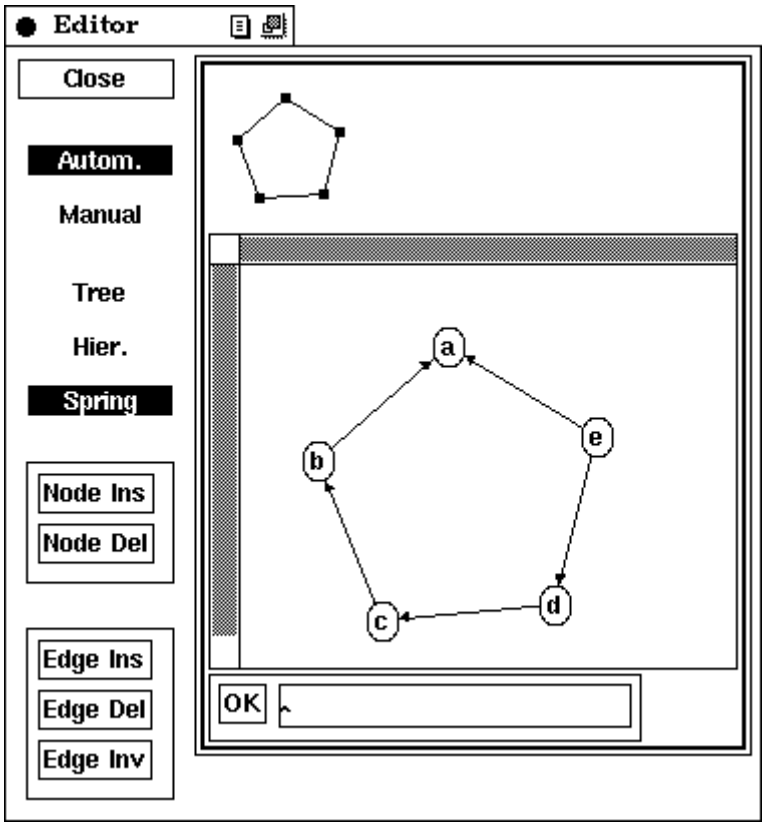

Figura 6: Grafo B desenhado pelo algoritmo de Kamada

Estes projectos, onde foram gerados grafos com várias centenas de nós, permitiram a elaboração de testes bastante exaustivos que foram importantes para detectar alguns problemas em aberto:

- ainda não são suportadas restrições definidas pelo utilizador (forma e localização de subgrafos),
- existem alguns problemas com grafos dinâmicos (alguns algoritmos não têm em conta o estado anterior do grafo),
- existem problemas no desenho de grafos com nós de diferentes tamanhos (especialmente no algoritmo de Kamada).

Para resolver o primeiro problema terá de ser feito trabalho na direcção do indicado por Böhringer em [BP90]. Uma parte da solução passará pela construção de um gestor de restrições. No entanto, este não é um problema de fácil solução.

Os grafos dinâmicos poderão ser melhor suportados se os algoritmos forem alterados de acordo com o trabalho de Messinger [Mes88], isto é, com estratégias de divisão e conquista.

Para resolver o problema do tamanho dos nós será necessário extender os algoritmos. Nomeadamente em relação ao algoritmo de Kamada uma possível

solução passará pela alteração do modo como o comprimento dos arcos entre os nós é calculado.

Ter-se-á, também, de realizar algum trabalho na direcção do apontado por Marks [Mar90] e Kosak [KMS90] em que é referida a necessidade de definir formalismos sintácticos e semânticos para grafos. Estes formalismos suportarão a definição de critérios estéticos pelo utilizador e serão os responsáveis pela sua passagem ao gestor de restrições.

Agradecimentos

Gostaria de agradecer ao Spa (Salvador P. Abreu) pelo seu suporte ao longo deste trabalho e ao Gabriel Lopes pelos seus comentários e sugestões.

Referências

[Abr89] Salvador Pinto Abreu. *ALPES X-Prolog Programming Manual.* CRIA/UNINOVA, 1989.

[BP90] Karl-Friedrich Boehringer and Frances Newbery Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *SIGCHI'90 Proceedings*, 1990.

[CNAO85] Norishige Chiba, Takao Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using pq-trees. *Journal Comput. System Sci.*, pages 54–76, 1985.

[CON85] Norishige Chiba, Kazunori Onoguchi, and Takao Nishizeki. Drawing plane graphs nicely. *Acta Informatica*, 22:187–201, 1985.

[ET89] Peter Eades and Roberto Tamassia. Algorithms for automatic graph drawing: An annotated bibliograph. Technical Report CS-89-09, University of Queensland, Dept. of Computer Science, St. Lucia, Queensland, 4067, Australia, October 1989.

[GNV88] E. R. Gansner, S. C. North, and K. P. Vo. DAG: A program that draws directed graphs. *Software - Practice and Experience*, 18(11), November 1988.

[JG89] David Jablonowski and Vincent A. Guarna. GMB: A tool for manipulating and animating graph data structures. *Software - Practice and Experience*, 19(3), March 1989.

[KK89] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[KMS90] Corey Kosak, Joe Marks, and Stuart Shieber. New approaches to automatic network-diagram layout. draft, 1990.

[Mar90] Joseph Marks. A syntax and semantics for network diagrams. In *Proceedings of the 1990 IEEE Workshop on Visual Languages*, pages 104–110, Illinois, October 1990.

[MAS88] Joel McCormack, Paul Asente, and Ralph R. Swick. *X Toolkit Intrinsics—C Language Interface*. MIT Project Athena, 1988. For X Version 11, Release 4.

[Mes88] Eli Benjamin Messinger. Automatic layout of large directed graphs. Technical Report 99-07-08, Department of Computer Science, Washington University, Seattle, Washington 98195, July 1988.

[Moe90] Sven Moen. Drawing dynamic trees. *IEEE Software*, pages 21–28, July 1990.

[New88a] Frances J. Newbery. EDGE: An extendible directed graph editor. Technical Report 8/88, Department of Informatics, University of Karlsruhe, 7500 Karlsruhe 1, West Germany, June 1988.

[New88b] Frances J. Newbery. An interface description language for graph editors. In *1988 IEEE Workshop on Visual Languages*, pages 144–149, Pittsburgh, PA, October 10–12 1988.

[Pet89] Chris D. Peterson. Athena widget set—c language interface. Technical report, MIT X Consortium, 1989. For X Version 11, Release 4.

[PPM91] J. Moura Pires, H. Pinheiro Pita, and L. Camarinha Matos. Specification of a platform for the development of graphical user interfaces. Technical report, DI/UNL, July 1991.

[Qua90] Paulo Quaresma. Grafos na representação e visualização de conhecimento. Technical report, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, September 1990.

[RDM+87] Lawrence A. Rowe, Michael Davis, Eli Messinger, Carl Meyer, Charles Spirakis, and Allen Tuan. A browser for directed graphs. *Software - Practice and Experience*, 17(1):61–76, January 1987.

[RT81] Edward M. Reingold and John S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, 7(2):223–228, March 1981.

[STT81] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(2):109–125, February 1981.

[TBB88] Roberto Tamassia, Guuseppe Di Battista, and Carlo Batini. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, SE-18(1):61–79, January/February 1988.

[Wer91] Michel Wermelinger. *GET: Graph Editor and Tools—The Incomplete Reference*. CRIA/UNINOVA, January 1991.