Sinergia em Desenho de Grafos Usando Springs e Pequenas Heurísticas

H. A. D. do Nascimento
hadn@dcc.unicamp.br
C. F. X. de Mendonça N.
xavier@dcc.unicamp.br
P. S. de Souza
pedros@vnet.ibm.com

Relatório Técnico IC–96-15

Outubro de 1996

Sinergia em Desenho de Grafos Usando Springs e Pequenas Heurísticas

H. A. D. do Nascimento*† hadn@dcc.unicamp.br
C. F. X. de Mendonça N.*† xavier@dcc.unicamp.br
P. S. de Souza‡§ pedros@vnet.ibm.com

Sumário

Este trabalho mostra que fazendo-se uso de "A-Teams" (Times Assíncronos) podemos combinar algoritmos como Springs, Baricentro e heurísticas aleatórias, de uma maneira simples e flexível, de modo que cooperem sinergeticamente permitindo desenho de grafos com duas características estéticas, cujos problemas correspondentes são conflitantes e NP-difíceis.

Abstract

In this paper we discuss how to make use of A-Teams (Asynchronous Teams) to combine algorithms like Springs, Barycenter and random heuristics in a flexible way such that they cooperate synergetically to draw graphs with two NP-hard conflicting aesthetics.

1 Introdução

O problema de desenhar grafos está inserido na sub-área de Geometria Computacional, e vem ganhando grande interesse nos últimos anos. Dentre os diversos problemas relativos à esta área de pesquisa, vamos nos concentrar no desenho de grafos gerais.

Comumente, estuda-se, em teoria dos grafos, como partir de uma formulação geométrica e chegar à uma classe de grafos. Dentro de desenho de grafos, desejamos trabalhar no sentido inverso. Dada uma classe de grafos, gostaríamos de obter uma geometria que expressasse visualmente esta classe. Embora aparentemente fácil, este problema tem se mostrado ser em geral NP-difícil, mesmo se considerado sob um único critério estético [9]. Citamos como exemplos os problemas de desenhar grafos não planares com mínimo número de cruzamentos de arestas [11] e mostrando o máximo de simetrias [9].

Outras dificuldades encontradas são: a questão da flexibilidade e do tempo de resposta dos processos tradicionais. Embora alguns processos tenham certas características desejáveis e bom tempo de resposta, como por exemplo Springs [7, 13], que conhecidamente são algoritmos que desenham grafos mostrando um grande número de simetrias, eles sofrem

*Instituto de Computação - UNICAMP, SP, Brasil
†Apoio financeiro: LAC-FAPESP, CNPq e FAEP
‡IBM Consulting Group, 3200 Windy Hill Road, IMA/WG09B, Atlanta, Ga 30339
§Apoio financeiro: LAC-FAPESP

a deficiência de falta de flexibilidade; em outras palavras, é impossível modificar os critérios de embelezamento destes processos por métodos tradicionais sem destruir suas principais características. Outro problema apresentado por estes sistemas é que eles são incapazes de reduzir cruzamentos [1, 7, 10]. Reduzir cruzamentos no desenho gerado por Springs é um assunto de grande interesse, mesmo para desenhar árvores [8]. O sistema aqui proposto cobre algumas destas fraquezas como ilustramos na seção 4.

Neste trabalho mostramos como reduzir cruzamentos de arestas e, ao mesmo tempo, conseguir as simetrias proporcionadas pelo sistema Springs. Tal flexibilidade é resultado da sinergia entre algoritmos que favorecem simetrias e pequenas heurísticas que diminuem o número de cruzamentos.

De um modo geral, quando confrontados com o problema de desenhar grafos com razoáveis características estéticas podemos tomar uma entre três escolhas:

- escolher o algoritmo mais conveniente e utilizá-lo para desenhar nosso grafo;
- desenhar nosso grafo com todos os algoritmos disponíveis e escolher o desenho mais conveniente;
- organizar os algoritmos disponíveis de modo a que cooperem entre si, produzindo desenhos com características desejáveis.

Na seção 3, mostramos que é possível combinar algoritmos baseados em Springs com heurísticas de redução de cruzamentos e com algoritmos aleatórios, a fim de obtermos as características estéticas desejadas.

2 Terminologia

Introduzimos aqui algumas definições para situar o leitor nos problemas que envolvem desenho de grafos e no conceito de Times Assíncronos.

2.1 Grafos

Um *grafo* $G = (V, E)$ consiste de um conjunto de vértices V e um conjunto de "arestas" E. Uma *aresta* é um par não ordenado de vértices distintos. (O leitor acostumado com estas definições notará que tratamos de grafos não orientados, simples, sem laços ou arestas múltiplas). Em uma aresta $e = uv$ dizemos que os vértices u e v são os extremos de e. Arestas distintas não podem ter ambos os extremos iguais.

Dois vértices u e v são *adjacentes* se existe uma aresta $e = uv$ em E; neste caso dizemos que a aresta e é incidente em u e em v. O *grau* d_v de um vértice v é a soma das arestas de G incidentes em v.

Um *caminho* entre os vértices u e v em um grafo $G = (V, E)$ é uma seqüência $C_{u,v} = (u = c_0, c_0c_1, c_1, c_1c_2, ..., c_{k-1}c_k, c_k = v)$ em G, que não repete vértices. O *comprimento* de um caminho é determinado pelo número de suas arestas. A *distância* entre dois vértices é determinada pelo caminho de menor comprimento.

Dizemos que um grafo G é *conexo* se para todo par de vértices u e v em G existe um caminho $C_{u,v}$ em G.

2.2 Desenho de Grafos

O *desenho* de um grafo $G = (V, E)$ é uma função $D_G : V \rightarrow I\!R^2$ que associa uma posição no plano a cada vértice v de V. Quando o grafo estiver subentendido, ele será omitido e utilizaremos a notação D.

A forma pictórica que adotamos para um grafo é desenhar os vértices como círculos e as arestas como linhas ligando dois círculos. Dentro desta forma pictórica temos claramente dois problemas: "roting" e "placement". O primeiro problema consiste em fixar os vértices e rotear as arestas do grafo convenientemente. O segundo consiste em adotar um padrão gráfico para desenhar as arestas e encontrar uma localização para os vértices. No nosso caso, damos enfoque ao problema de "placement", adotando um padrão gráfico em que arestas são linhas retas e procurando uma localização para os vértices do grafo, de modo que os desenhos obtidos satisfaçam alguns critérios estéticos.

Em especial, os critérios estéticos que desejamos alcançar são dois:

- apresentar o maior número de simetrias e
- possuir mínimo número de cruzamentos de arestas.

Caracterizamos simetrias no sentido informal pela facilidade de seu reconhecimento visual, tais como: reflexões, rotações, isomorfismos de grafos ou uma combinação destas propriedades.

2.3 Times Assíncronos

A primeira organização de "Times Assíncronos", ou *A-Teams* (do inglês "Asynchronous Teams"), foi apresentada por Talukdar e Souza em [2, 3, 4, 19]. Esta organização constitui-se de algoritmos (agentes) que trabalham de forma **assíncrona** e **cíclica** sobre um conjunto de dados depositados em memórias compartilhadas. Os agentes participantes da organização freqüentemente possuem características distintas e geram, portanto, resultados diferentes. Contudo, quando integrados, estes mesmos agentes formam um verdadeiro "time" (ou uma equipe) capaz de alcançar resultados melhores do que cada um deles isoladamente.

Nos Times Assíncronos os algoritmos são autônomos, não havendo um agente supervisor que controla ou influencia a execução dos demais. Cada agente executa de maneira independente e contribui com o seu trabalho para a resolução de um problema comum (que pode envolver funções multi-objetivos). Além disso, os resultados obtidos por um agente são armazenados nas memórias, ficando disponíveis para serem reutilizados por outros algoritmos. Isto proporciona a cooperação entre os agentes, a qual aumenta as chances do time encontrar melhores soluções (sinergia).

As memórias, nos Times Assíncronos, armazenam soluções parciais que são gradativamente melhoradas. Como memória é um recurso limitado, algumas soluções precisam ser removidas para que novas soluções sejam inseridas. Esta tarefa é deixada a cargo de um agente denominado *agente destruidor de soluções*, que seleciona e remove soluções da memória, segundo uma "política de destruição". Este agente é um elemento crítico de time. É ele quem determina quais soluções devem permanecer e quais devem ser eliminadas por serem pouco promissoras.

Como exemplo, a figura 1 mostra a organização de um Time Assíncrono genérico, composto de 1 memória e 4 agentes. A memória está simbolizada por um retângulo e os agentes estão representados por setas. A direção das setas indica de onde os agentes lêem soluções e onde eles escrevem as novas soluções.

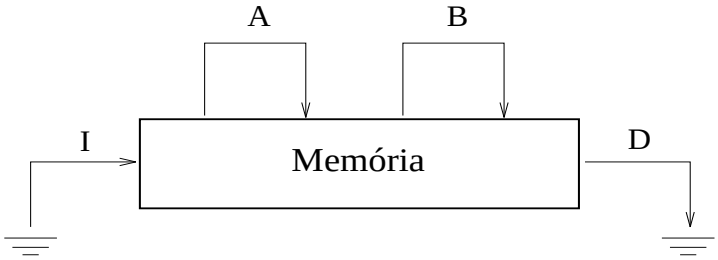


Figura 1: Representação gráfica de um A-Team

O agente **I** é chamado de *agente iniciador da memória*. Ele é responsável por gerar as soluções iniciais. O agente **D** é um destruidor de soluções e os agentes **A** e **B** compõem novas soluções partindo de soluções já existentes.

Devido as características da organização do A-Team, isto é, a não existência de um supervisor e nem da necessidade de sincronismo na comunicação entre agentes para que eles cooperem, um Time Assíncrono apresenta um alto grau de paralelismo e é adequado para o processamento distribuído [2, 15].

3 O Sistema

O nosso sistema consiste de um Time Assíncrono para desenho de grafos gerais que reune os seguintes agentes: Springs, buscando simetrias, e heurísticas que tentam reduzir o número de cruzamentos de arestas.

O algoritmo Springs [7, 13] basea-se na identificação de um grafo como um sistema dinâmico de molas no plano, em que cada vértice é descrito como uma partícula e cada par de vértices está conectado por uma mola. O comprimento e a constante elástica da mola dependem da distância entre os vértices correspondentes no grafo. Um método de otimização é aplicado para encontrar um estado de baixa energia do sistema. Os estados de baixa energia estariam associados a desenhos aceitáveis. O algoritmo trabalha basicamente como segue: o vértice mais "crítico" (que contribui com a maior parcela de energia do sistema) é movido para uma posição que minimiza a energia total do sistema; os demais vértices são mantidos na sua posição original. Esse procedimento é, então, repetido sucessivamente até que não haja mais vértices que contribuam com valores "altos" à função de energia.

Como mencionamos anteriormente, o Springs gera desenhos bastante agradáveis com muitas simetrias. Contudo, ele não trata o problema de minimizar o número de cruzamentos de arestas, e muitos ajustes devem ser feitos para que o sistema não caia em um mínimo local [14].

Visando superar esta deficiência, combinamos, através de um A-Team, o Springs com heurísticas baseadas no método do Baricentro descrito por Sugiyama [6, 17, 18] e com

agentes aleatórios que movem os vértices para posições quaisquer de uma região do plano.

3.1 Estrutura do Time Assíncrono

O Time assíncrono desenvolvido possui 8 agentes e 1 memória, como ilustra a figura 2.

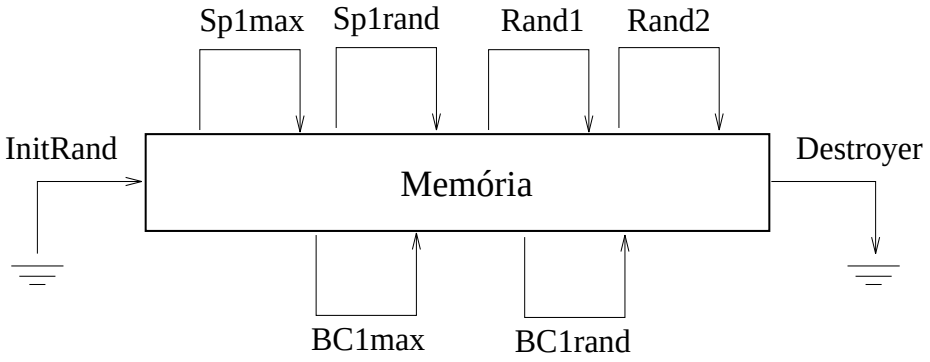


Figura 2: A-Team para o desenho de grafos

Denominamos de *soluções*, os desenhos produzidos pelos agentes. Os primeiros desenhos gerados são chamados de *soluções iniciais*.

A memória do nosso sistema contém 100 *células*, cada uma capaz de armazenar uma solução (coordenadas X e Y reais dos vértices de um grafo). A memória é de *acesso aleatório* e as operações definidas sobre a mesma são de *leitura* e de *escrita*. Em geral, o número de células da memória é proporcional ao tamanho do grafo que se pretende desenhar.

Os agentes que integram o Time Assíncronos são detalhados a seguir:

InitRand agente iniciador. Soluções são construídas escolhendo-se coordenadas X e Y aleatórias dentro de uma região delimitada do plano para todos os vértices do grafo. As soluções geradas são armazenadas na memória.

Sp1max versão reduzida do algoritmo Springs proposto por Kamada [13]. O agente Sp1max lê uma solução da memória, move o vértice mais crítico da mesma para uma posição de baixa energia e escreve a nova solução na memória.

Sp1rand é semelhante ao agente Sp1max, exceto pelo fato de que escolhe um vértice ao acaso para ser movido.

BC1max lê uma solução, descobre o vértice que contribui com o maior número de cruzamento de arestas e move-o para o seu baricentro (calculado a partir das coordenadas X e Y de seus adjacentes). A solução resultante é escrita na memória.

BC1rand semelhante ao agente BC1max, contudo, move para seu baricentro um vértice escolhido ao acaso.

Rand1 toma um vértice qualquer e move-o para uma posição escolhida aleatoriamente dentro de um região retangular que contenha o grafo. Escreve a solução na memória.

Rand2 semelhante ao Rand1, exceto pelo fato de que o vértice é movido dentro uma pequena vizinhança.

Destroyer agente destruidor de soluções. Este agente remove uma solução da memória sempre que uma nova solução precisa ser inserida. A *política de destruição* que utilizamos adota uma distribuição "linear" de probabilidades em que as piores soluções têm grandes chances de serem eliminadas. A forma de medir a qualidade de uma solução será discutida posteriormente.

O primeiro agente do Time Assíncrono a ser executado é o iniciador InitRand.

Em seguida, os demais agentes entram no time. Estes agentes escolhem aleatoriamente as células da memória que serão lidas e trabalham autônoma e iterativamente sobre cópias de seus conteúdos, produzindo novas soluções.

O Time Assíncrono permanece em execução até que nenhuma mudança significativa, em termos de simetria e número de cruzamentos, seja feita no conjunto das melhores soluções obtidas. Neste momento, a memória deve conter um conjunto de boas soluções que satisfazem, em algum termo, os critérios de máxima simetria e de menor número de cruzamento de arestas.

Na seção seguinte, apresentaremos os conceitos básicos usados para quantificar e avaliar a qualidade das soluções.

3.2 Vetor de Qualidade das Soluções

Seja D_G uma solução (desenho) de um grafo $G = (V, E)$ no conjunto de todas as possíveis soluções $\mathcal{D}$ de G. Seja também $q : \mathcal{D}_G \rightarrow I\!R^2$ uma função que associa a cada solução um par de valores reais. Cada valor quantifica a penalidade da não satisfação de um critério estético. Quanto maior este valor, pior é a solução no critério correspondente. A primeira coordenada contém o valor da energia do Springs; esta é uma aproximação do problema de encontrar simetrias. A segunda coordenada contém o número de cruzamentos de arestas.

Dizemos, então, que $q(D_G) = (d_1, d_2)$ é o *vetor de qualidade* da solução D_G.

Para avaliar as soluções, definimos a seguir o conceito de *dominância* aplicado ao Time Assíncrono [5, 16].

dominância de soluções

Sejam D'_G e D''_G duas soluções (desenhos) de um grafo $G = (V, E)$. Sejam também $q(D'_G) = (d'_1, d'_2)$ e $q(D''_G) = (d''_1, d''_2)$ os vetores de qualidade correspondentes às soluções. Nós dizemos que a solução D' *domina* a solução D'' se $d'_i \leq d''_i$ para todo $i = 1, 2$ e $d'_k < d''_k$ para algum k.

Uma *camada* de soluções consiste de um conjunto de soluções não dominantes entre si. Estas soluções são consideradas *equivalentes*. Uma camada $\mathcal{C}_1$ *domina* uma camada $\mathcal{C}_2$ se pelo menos uma solução de $\mathcal{C}_1$ domina uma solução de $\mathcal{C}_2$. Neste caso dizemos que a camada $\mathcal{C}_1$ é mais *baixa* do que a camada $\mathcal{C}_2$. A camada de soluções mais baixa entre todas as possíveis camadas de soluções para o problema define o conjunto de soluções *eficientes*.

Definido o modo de avaliar soluções podemos ordená-las pela ordem das camadas a que pertencem. Desta forma, o agente destruidor pode facilmente determinar quais são as soluções mais ou menos "promissoras".

Outro resultado interessante da manipulação de camadas, é que o Time Assíncrono consegue produzir muito mais que uma única solução por execução. Em verdade, ele encontra um conjunto de soluções equivalentes (camada) que satisfaz os critérios estéticos desejados em diferentes proporções.

3.3 Detalhes de Implementação

Há várias formas de implementar um A-Team, dentre as quais destacamos:

1. **procedimentos de um programa**: os agentes são implementados como procedimentos de um programa simples. Uma das vantagens desta abordagem é a sua fácil codificação. Entretanto, torna-se extremamente difícil executar em paralelo os agentes possivelmente concorrentes.

2. **threads**: os agentes são implementados como threads. Esta abordagem é melhor do que a anterior. Ao custo de uma codificação um pouco mais complexa, ela permite paralelismo em uma sistema operacional multi-thread executando sobre uma máquina paralela.

3. **processos independentes**: os agentes são implementados como programas independentes que acessam uma memória compartilhada. A codificação não é tão fácil como nas outras implementações, contudo, os processos podem ser executados em um sistema distribuído sobre uma rede de computadores, alcançando, conseqüentemente, alto grau de paralelismo.

Neste trabalho, adotamos a terceira abordagem frente às suas vantagens. O Time Assíncrono foi implementado e testado em uma rede de computadores. Entre as características mais importantes do sistema citamos: a facilidade de adicionar um novo agente ao time, e o grande potencial de se explorar paralelismo replicando agentes em diversas máquinas.

4 Resultados

Nesta seção apresentaremos alguns desenhos obtidos pelo nosso sistema.

A figura 3(a) mostra o desenho de uma solução inicial, produzida pelo agente InitRand. As soluções iniciais são bastante aleatórias, o que proporciona uma grande diversidade de desenhos para serem escolhidos e melhorados.

As figuras 3(b) e 3(c) são desenhos do mesmo grafo e foram obtidos pelo time de agentes a partir das soluções iniciais.

A figura 3(b) é um desenho típico gerado por algoritmos Springs. Sua disposição mostra simetrias e expressa a topologia tridimensional do grafo. Contudo, na figura 3(c), temos um desenho que é impossível de ser produzido pelos agentes Springs sozinhos. De fato, este desenho, que apresenta um alto valor de energia no sistema de molas, só foi conseguido por que há sinergia entre os diversos algoritmos incorporados ao Time Assíncrono.

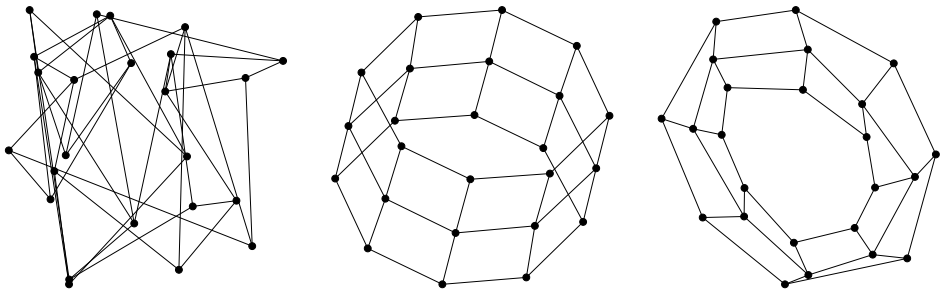


Figura 3:

Observe, agora, seis desenhos do grafo completo C9 na figura 4. Todos eles fazem parte da melhor camada, a camada mais baixa na memória, e foram obtidos após algum tempo de execução do sistema. Os desenhos 4(a), 4(b), 4(c), 4(d), 4(e) e 4(f) estão apresentados em ordem decrescente de simetrias e possuem, respectivamente, 106, 94, 68, 52, 48 e 36 cruzamentos.

Este exemplo ilustra muito bem como os critérios estéticos podem ser conflitantes. Na medida em que reduzimos o número de cruzamentos, perdemos simetrias.

Apesar disso, o Time Assíncrono conseguiu um amplo espectro de boas soluções.

Na figura 5 vemos desenhos de um cubo. Observe que o desenho 5(b) possui algumas aresta muito próximas, quase resultando em cruzamentos. Analisando o comportamento do time, percebemos que os agentes Springs tendem a empurrar os vértices internos deste desenho para fora, tentando reorganizá-los como fez na solução apresentada na figura 5(a), que tem mínima energia. Porém, qualquer movimento de apenas um vértice por vez produz soluções com muitos cruzamentos e alta energia inicial, as quais terminam por serem eliminadas da memória. Deste modo, tudo o que os agentes Springs conseguem fazer é aproximar estes vértices internos das arestas que delimitam externamente o desenho, reduzindo a energia em pequenos níveis, mas sem gerar novos cruzamentos.

Em função dessa característica, alguns desenhos com poucos cruzamentos podem não

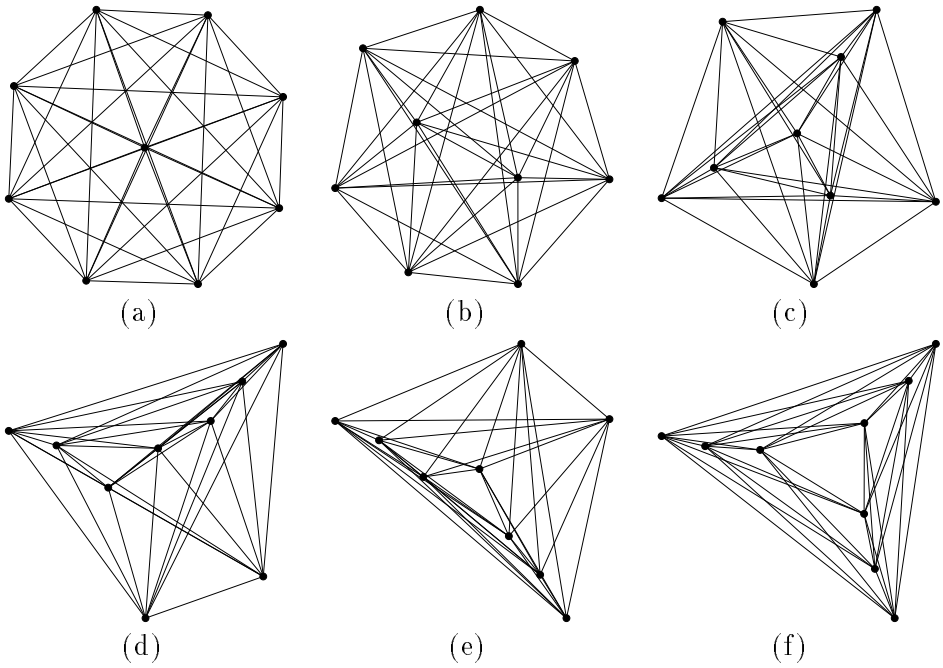


Figura 4:

ficar simétricos, embora haja uma forma de representá-los com mais simetrias. Veja, por exemplo, a figura 5(c). Infelizmente, não conseguimos obter esta solução, pois os seus vértices têm energias muito altas.

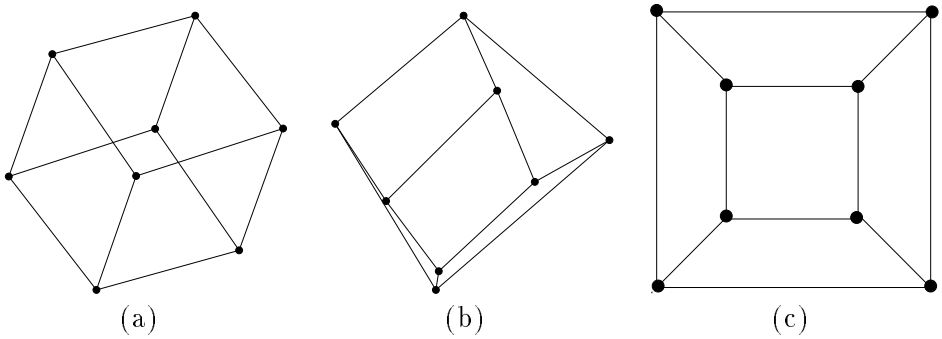
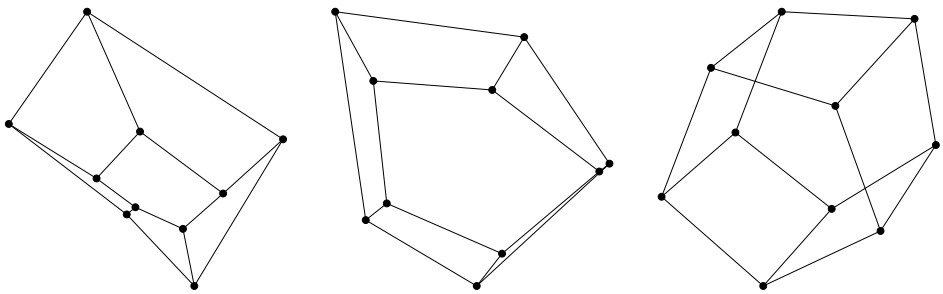


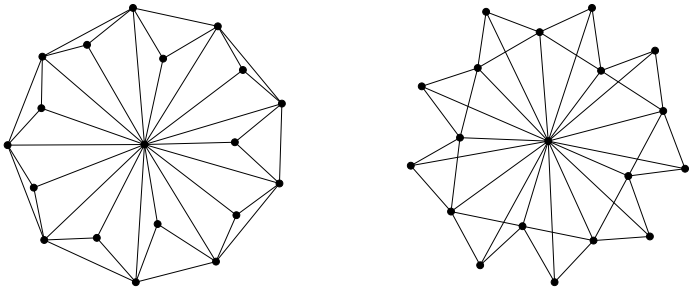
Figura 5:

As figuras 6 e 7 mostram outros desenhos que conseguimos com o nosso sistema.



(a) (b) (c)

Figura 6:



(a) (b)

Figura 7:

5 Conclusão

Os agentes do nosso Time Assíncrono cooperam entre si e por isso conseguem soluções que aproveitam a simetria implícita nos algoritmos Springs e a redução de cruzamentos proporcionada por heurísticas simples como o Baricentro.

O sistema é muito interessante, porque resulta em não apenas uma, mas muitas soluções finais equivalentes, de acordo com o grafo. Além disso, novos algoritmos que trabalham sobre simetrias ou número de cruzamentos podem seu adicionados ao time a fim de aumentar a sinergia entre os agentes.

A idéia de utilizar A-Teams para desenho grafos é extremamente flexível, podendo ser estendida para trabalhar sobre outros critérios estéticos, sejam eles conflitantes ou não. É possível, por exemplo, adicionar o critério de melhor resolução angular [12], visando obter uma distribuição mais uniforme dos ângulos nos desenhos com poucos cruzamentos.

Por fim, a idéia pode ser aplicada a outras classes de grafos, sendo que pesquisas nesse sentido já estão sendo realizadas pelo grupo dos autores.

Referências

[1] R. Davidson and D. Harel. Drawing Graphs Nicely Using Simulated Annealing. Technical Report CS89-13, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, 1989.

[2] P. S. de Souza. *Organizations for Multi-Algorithm Problems.* Ph.D. dissertation, Department of Electrical and Computer Engineering of Carnegie Mellon University, April 1993.

[3] P. S. de Souza and S. N. Talukdar. Genetic Algorithms in Asynchronous Teams. In *Proc. of the Fourth International Conference on Genetic Algorithms*, Los Altos, CA, 1991.

[4] P. S. de Souza and S. N. Talukdar. Asynchronous Organizations for Multi-Algorithm Problems. In *Proc. ACM Symposium on Applied Computing*, Indianapolis, IN, February 1993.

[5] R. F. Rodrigues e P. S. de Souza. Asynchronous Teams: A Multi-Algorithm Approach for Solving Combinatorial Multi-Objective Optimization Problems. In *Proceedings of the 5th Workshop of the DGOR-Working Group Multicriteria Optimization and Decision Theory*, Germany, May 1995.

[6] P. Eades and K. Sugiyama. How to draw a directed graph. *Journal of Information Processing*, pages 424–437, 1991.

[7] P. D. Eades. A Heuristic for Graph Drawing. *Congr. Numer.*, 42:149–160, 1984.

[8] P. D. Eades. Drawing free trees. *Bulletin of the Institute for Combinatorics and its Applications*, 5:10–36, 1992.

[9] P. D. Eades and R. Tamassia. Algorithms for Drawing Graphs: an Annotated Bibliography. Technical Report CS-89-09, Department of Computer Science, Brown University, 1989.

[10] T. Fruchterman and E. Reingold. Graph Drawing by Force-Directed Placement. *Software – Practice and Experience*, 21(11):1129–1164, 1991. also as Technical Report UIUCDCS-R-90-1609, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 1990.

[11] M. R. Garey and D. S. Johnson. Crossing Number is NP-Complete. *SIAM J. Algebraic Discrete Methods*, 4(3):312–316, 1983.

[12] A. Garg. On Drawing Angle Graphs. *Lecture Notes in Computer Science (Proc. DIMACS International Workshop Graph Drawing, 1994)*, 894:84–95, 1995.

[13] T. Kamada and S. Kawai. Automatic Display of Network Structures for Human Understanding. Technical Report 88-007, Department of Information Science Faculty of Science, University of Tokyo, Tokyo, 1988.

[14] X. Lin. *Analysis of Algorithms for Drawing Graphs.* PhD thesis, University of Queensland, Department of Computer Science, University of Queensland, 1992.

[15] H. P. Peixoto. Metodologia de Especificação de Times Assíncronos para Problemas de Otimização Combinatória. M.Sc. thesis, Instituto de Matemática, Estatística e Ciência da Computação, Universidade Estadual de Campinas, Campinas - SP, Brazil, 1995.

[16] R. F. Rodrigues. Times Assíncronos para a Resolução de Problemas de Otimização Combinatória Multiobjetivo. M.Sc. thesis, Instituto de Computação, Universidade Estadual de Campinas, Campinas - SP, Brazil, in print 1996.

[17] K. Sugiyama and K. Misue. Visualization of Structural Information: Automatic Drawing of Compound Digraphs. *IEEE Transactions on Software Engineering*, 21(4):876–892, 1991.

[18] K. Sugiyama, S. Tagawa, and M. Toda. Methods for Visual Understanding of Hierarchical Systems. *IEEE Trans. Syst. Man Cybern.*, SMC-11(2):109–125, 1981.

[19] S. N. Talukdar and P. S. de Souza. Asynchronous Teams. In *Proc. Second Siam Conf. on Linear Algebra: Signals, Systems, and Control*, San Francisco, CA, Nov. 1990.

Relatórios Técnicos – 1995

95-01 **Paradigmas de algoritmos na solução de problemas de busca multidimensional,** *Pedro J. de Rezende, Renato Fileto*

95-02 **Adaptive enumeration of implicit surfaces with affine arithmetic,** *Luiz Henrique de Figueiredo, Jorge Stolfi*

95-03 **W3 no Ensino de Graduação?,** *Hans Liesenberg*

95-04 **A greedy method for edge-colouring odd maximum degree doubly chordal graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

95-05 **Protocols for Maintaining Consistency of Replicated Data,** *Ricardo Anido, N. C. Mendonça*

95-06 **Guaranteeing Full Fault Coverage for UIO-Based Methods,** *Ricardo Anido and Ana Cavalli*

95-07 **Xchart-Based Complex Dialogue Development,** *Fábio Nogueira de Lucena, Hans K.E. Liesenberg*

95-08 **A Direct Manipulation User Interface for Querying Geographic Databases,** *Juliano Lopes de Oliveira, Claudia Bauzer Medeiros*

95-09 **Bases for the Matching Lattice of Matching Covered Graphs,** *Cláudio L. Lucchesi, Marcelo H. Carvalho*

95-10 **A Highly Reconfigurable Neighborhood Image Processor based on Functional Programming,** *Neucimar J. Leite, Marcelo A. de Barros*

95-11 **Processador de Vizinhança para Filtragem Morfológica,** *Ilka Marinho Barros, Roberto de Alencar Lotufo, Neucimar Jerônimo Leite*

95-12 **Modelos Computacionais para Processamento Digital de Imagens em Arquiteturas Paralelas,** *Neucimar Jerônimo Leite*

95-13 **Modelos de Computação Paralela e Projeto de Algoritmos,** *Ronaldo Parente de Menezes e João Carlos Setubal*

95-14 **Vertex Splitting and Tension-Free Layout,** *P. Eades, C. F. X. de Mendonça N.*

95-15 **NP-Hardness Results for Tension-Free Layout,** *C. F. X. de Mendonça N., P. Eades, C. L. Lucchesi, J. Meidanis*

95-16 **Agentes Replicantes e Algoritmos de Eco,** *Marcos J. C. Euzébio*

95-17 **Anais da II Oficina Nacional em Problemas Combinatórios: Teoria, Algoritmos e Aplicações,** *Editores: Marcus Vinicius S. Poggi de Aragão, Cid Carvalho de Souza [Not available.]*

95-18 **Asynchronous Teams: A Multi-Algorithm Approach for Solving Combinatorial Multiobjective Optimization Problems,** *Rosiane de Freitas Rodrigues, Pedro Sérgio de Souza*

95-19 **wxWindows: Uma Introdução,** *Carlos Neves Júnior, Tallys Hoover Yunes, Fábio Nogueira de Lucena, Hans Kurt E. Liesenberg*

95-20 **John von Neumann: Suas Contribuições à Computação,** *Tomasz Kowaltowski*

95-21 **A Linear Time Algorithm for Binary Phylogeny using PQ-Trees,** *J. Meidanis and E. G. Munuera*

95-22 **Text Structure Aiming at Machine Translation,** *Horacio Saggion and Ariadne Carvalho*

95-23 **Cálculo de la Estructura de un Texto en un Sistema de Procesamiento de Lenguaje Natural,** *Horacio Saggion and Ariadne Carvalho*

95-24 **ATIFS: Um Ambiente de Testes baseado em Inje,c ao de Falhas por Software,** *Eliane Martins*

95-25 **Multiware Plataform: Some Issues About the Middleware Layer,** *Edmundo Roberto Mauro Madeira*

95-26 **WorkFlow Systems: a few definitions and a few suggestions,** *Paulo Barthelmess and Jacques Wainer*

95-27 **Workflow Modeling,** *Paulo Barthelmess and Jacques Wainer*

Relatórios Técnicos – 1996

96-01 **Construção de Interfaces Homem-Computador: Uma Proposta Revisada de Disciplina de Graduação,** *F'abio Nogueira Lucena and Hans K.E. Liesenberg*

96Abs **DCC-IMECC-UNICAMP Technical Reports 1992–1996 Abstracts,** *C. L. Lucchesi and P. J. de Rezende and J.Stolfi*

96-02 **Automatic visualization of two-dimensional cellular complexes,** *Rober Marcone Rosi and Jorge Stolfi*

Instituto de Computação
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

reltec@dcc.unicamp.br