Uma Abordagem para Desenho de Grafos Baseada na Utilização de Times Assíncronos

Hugo Alexandre Dantas do Nascimento

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Uma Abordagem para Desenho de Grafos Baseada na Utilização de Times Assíncronos

Hugo Alexandre Dantas do Nascimento[1]

Maio de 1997

Banca Examinadora:

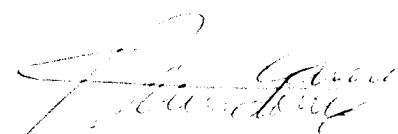
- C. F. Xavier de Mendonça N.
  DTC - IC - UNICAMP (Orientador)
- Cid Carvalho de Souza
  Instituto de Computação - UNICAMP
- Marcus Vinicius Soledade Poggi de Aragão
  Departamento de Informática - PUC-Rio
- Jorge Stolfi (Suplente)
  Instituto de Computação - UNICAMP

[1]Apoio Financeiro da FAPESP, CNPq e FAEP.

Uma Abordagem para Desenho de Grafos Baseada na Utilização de Times Assíncronos

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Hugo Alexandre Dantas do Nascimento e aprovada pela Banca Examinadora.

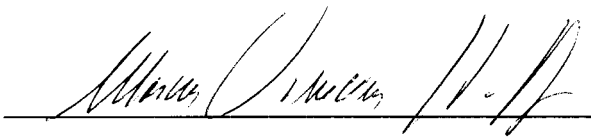
Campinas, 9 de maio de 1997.



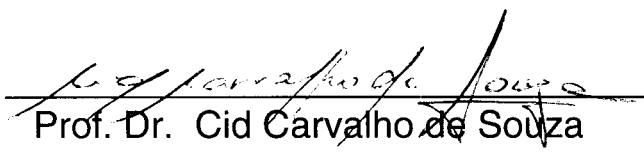
C. F. Xavier de Mendonça N. DTC - IC - UNICAMP
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

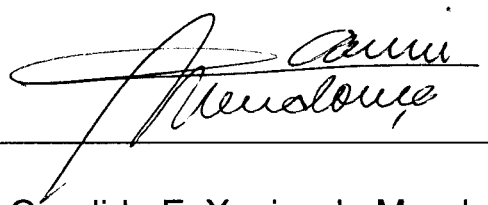
Tese de Mestrado defendida e aprovada em 05 de maio de 1997 pela Banca Examinadora composta pelos Professores Doutores



Prof. Dr. Marcus Vinícius Poggi Aragão



Prof. Dr. Cid Carvalho de Souza



Prof. Dr. Cândido F. Xavier de Mendonça Neto

Dedico este trabalho aos meus pais e às minhas irmãs pelo amor e incentivo dispensados durante estes dois anos de lutas.

Agradecimentos

Todo trabalho, por mais simples que seja, sempre recebe as contribuições e o esforço de muitas pessoas. A presente dissertação não foge a regra: inúmeras foram as "mãos" que ajudaram na escrita das páginas que se seguem.

Inicialmente, agradeço o apoio constante de meus pais e de minhas irmãs. Os seus conselhos permaneceram sempre nítidos, reanimando-me a seguir no caminho que escolhi.

A meu orientador, Candido Xavier, por ter me apresentado o tema da dissertação e pela amizade sincera com que me auxiliou durante todo o curso.

A Pedro Sérgio de Souza, meu co-orientador, pelas conversas interessantes sobre a filosofia que está por trás da aplicação de Times Assíncronos, pela paciência em ouvir as minhas idéias e pelas valiosas sugestões ao trabalho.

Aos meus amigos de Campinas, Sr. José Herculano, D. Maria Dias, Rose, Douglas, Sr. Oswaldo Veçoso, D. Ignês Veçoso, Sr. Antônio Botelho, D. Maria José Botelho, D. Benedita e D. Lupcínia, que hoje constituem a minha segunda família.

Aos meus amigos de república, Raniere Vidal, Danúzia Pinto e Maria de Jesus Carneiro, que me ensinaram a ser paciente frente às diferenças humanas, e a saber ouvir bem mais do que saber falar.

Aos amigos de Natal que também vieram fazer o mestrado na Unicamp, entre eles: Aluízio Neto, Guilherme Queiroz, Agostinho Junior e Marco Antônio, por terem criado um "clima" natalense, apesar do frio e do calor extremos de Campinas.

Aos professores da Unicamp e aos colegas de curso, pelo grande esforço em desenvolver um ambiente de pesquisa rico em troca de conhecimentos.

E finalmente à FAPESP, ao CNPq e à FAEP pelo apoio financeiro sem o qual não seria possível realizar este trabalho. Agradeço ainda à FAPESP pela disponibilidade de um fundo de reserva técnica, que foi utilizado para a obtenção dos resultados finais da pesquisa.

O meu muito obrigado.

Resumo

Desenho de Grafos é uma área recente que trata do desenvolvimento de técnicas e de algoritmos para construir representações geométricas de grafos, atendendo, em geral, a critérios estéticos.

A atividade de desenhar grafos implica em muitas dificuldades; entre elas, verificamos que a satisfação de alguns critérios estéticos envolve freqüentemente problemas *NP-difíceis* e que, em muitos casos, os critérios são conflitantes entre si. Em função disso, heurísticas têm sido desenvolvidas e amplamente utilizadas para obter bons desenhos.

No presente trabalho, descrevemos uma nova abordagem para desenhar grafos, que se baseia na combinação de heurísticas utilizando um tipo de organização de agentes conhecido como Time Assíncrono.

A abordagem é capaz de produzir desenhos melhores do que as heurísticas isoladas, e é flexível pois pode ser aplicada para trabalhar com muitos critérios estéticos e com várias classes de grafos.

Abstract

Graph Drawing is a new area that deals with the development of techniques and algorithms whose major concern is the geometric representations of graphs. These geometric representations must follow a set of aesthetic criteria in a "nice" way.

The activity of drawing graphs run into many dificulties, for example: the problem of satisfying some aesthetic criteria is often *NP-hard* and, in most cases, there are some conflitcs among the criteria. This justifies the wide use of heuristics to produce good drawings.

In this work we show a new approach to draw graphs. This approach focuses on the combination of different heuristics in a specific organization of agents, called Asynchronous Team.

The approach achieves better drawings than the heuristics alone, and it can be applied to work with several aesthetic criteria for drawing many classes of graphs.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Desenho de Grafos: Importância e Aplicações

Grafos são estruturas simples, formadas por um conjunto de vértices e um conjunto de arestas utilizados para representar a relação entre objetos físicos ou abstratos em diversas áreas do conhecimento humano. O uso de grafos, muitas vezes, está associado à necessidade de expressá-los visualmente a fim de que possamos identificar e compreender as relações entre seus elementos. Entretanto, a visualização de um grafo não consiste apenas em exibir os seus vértices e arestas; é preciso também que o desenho produzido seja "legível". Em geral, considera-se que um desenho de um grafo é *legível* quando ele atende a determinados critérios estéticos tais como "simetria", distribuição uniforme dos vértices em uma dada região e poucos cruzamentos de arestas. Algoritmos para obter desenhos com essas características estéticas têm sido desenvolvidos e são estudados dentro de uma área de pesquisa denominada Desenho de Grafos.

A área de Desenho de Grafos é bastante emergente e se preocupa com o desenvolvimento de técnicas e de algoritmos para construir representações geométricas de grafos visando satisfazer a certos critérios, em especial a critérios estéticos. As técnicas em Desenho de Grafos têm inúmeras aplicações, sendo empregadas, por exemplo, em Micro-eletrônica para desenho de circuitos VLSI; em Engenharia de Software para representar estruturas modulares de programas e da hierarquia entre classes de objetos (principalmente nas linguagens visuais e nas ferramentas de desenvolvimento de sistemas); em aplicações de CAD para facilitar a análise e a manipulação de dados; e ainda em sistemas de banco de dados e nas interfaces visuais. Em alguns desses ítens, basta abrir um livro que discorra sobre os mesmos para perceber que as figuras ou diagramas ilustrativos têm um bom trabalho subjacente em desenho estético.

A crescente utilização de computadores e a adoção das interfaces gráficas como forma eficiente de interação Homem-Máquina têm despertado muito interesse pela área de Desenho de Grafos, concorrendo para um amplo uso dos algoritmos disponíveis e para o desenvolvimento de novas técnicas e ferramentas [DETT94]. Atualmente, estudos sobre Desenho de Grafos são realizados em diversos ramos da Computação, envolvendo tópicos em Teoria dos Grafos, Geometria Computacional, Complexidade de Algoritmos, Banco de Dados e Interfaces.

Apesar do grande interesse, inúmeras são as dificuldades que estão relacionadas com a tarefa de desenhar um grafo, como veremos a seguir.

1.2 Dificuldades em Desenho de Grafos

Uma das maiores dificuldades em Desenho de Grafos é obter desenhos bons ou quase ótimos em alguns critérios estéticos, tais como menor número de cruzamentos de arestas e número máximo de simetrias, os quais em geral são problemas *NP-difíceis* [DETT94, Ead89]. Além disso, os critérios são freqüentemente conflitantes entre si, não sendo possível satisfazer simultaneamente dois ou mais deles para uma dada classe de grafos.

Uma outra dificuldade na área deve-se à subjetividade na definição do que seja um desenho legível. Para algumas pessoas, um desenho é legível quando apresenta poucos cruzamentos, enquanto que para outras isto ocorre quando ele exibe simetrias. Uma vez que a definição de qualidade de um desenho é um fator de interesse para o usuário, é comum deixar a ele a tarefa de escolher critérios estéticos. É preciso, no entanto, que existam algoritmos flexíveis, que possam ser ajustados a fim de produzirem desenhos de acordo com os critérios subjetivos escolhidos. O grande problema está exatamente no fato de que a maioria dos algoritmos para desenho encontrados na literatura apresentam pouca ou nenhuma flexibilidade, salvo raras excessões como algumas abordagens baseadas em **Simulated Annealing** e **Algoritmos Genéticos**. Mesmo assim, tais abordagens inserem novas dificuldades relativas à complexidade de se ajustar os seus parâmetros de convergência, além de exigirem um elevado tempo computacional. Citamos como exemplo o trabalho de Mendonça em [dMN94], onde os parâmetros de um Simulated Annealing são ajustados automaticamente inferindo a estética do usuário, porém, em detrimento do tempo de execução; esse trabalho demonstra a dificuldade de se configurar um sistema flexível para desenho de grafos.

A complexidade dos problemas em Desenho de Grafos e a existência de conflitos entre os critérios estéticos têm justificado o desenvolvimento de muitos algoritmos baseados em estratégias heúristicas para encontrar soluções aproximadas. É desejável, contudo, que os algoritmos não apenas produzam boas soluções, mas também que sejam flexíveis, a fim de que possam ser aplicados a várias classes de grafos ou que atendam a um conjunto de diferentes critérios estéticos.

Na próxima seção, sugerimos uma nova abordagem para desenhar grafos que se mostra flexível e que produz, em alguns casos, desenhos melhores os algoritmos encontrados na literatura.

1.3 A Utilização de Times Assíncronos

Os Times Assíncronos são organizações formadas por vários agentes que se comunicam de maneira assíncrona através de memórias compartilhadas, e que trabalham iterativamente originando fluxos cíclicos de dados. Os Times Assíncronos têm sido empregados, principalmente, na área de otimização combinatória para unir heurísticas diferentes, fazendo-as cooperar com o objetivo

de conseguirem resultados próximos do ótimo. Aproveitando a idéia de combinar heurísticas, buscou-se, de modo semelhante, integrar algoritmos em Desenho de Grafos para obter desenhos com melhor qualidade estética.

De fato, as estratégias heurísticas disponíveis para desenhar grafos possuem suas qualidades e deficiências. Algumas heurísticas geram bons desenhos para determinados critérios estéticos, enquanto que outras mostram-se melhores em outros aspectos. Mesmo para um critério estético específico, as heurísticas podem apresentar desempenho variável de acordo com o grafo a ser desenhado. Assim, a idéia de unir algoritmos diferentes é uma forma de superar as dificiências de cada heurística.

A proposta original que deu início ao presente trabalho consistia em reunir heurísticas em um Time Assíncrono para desenhar Grafos Direcionados Acíclicos, satisfazendo a poucos critérios estéticos tais como: mostrar o mínimo de cruzamentos de arestas e apresentar arestas tão retas quanto possível. No entanto, foi observado que o uso de Times Assíncronos não apenas produzia desenhos melhores do que as heurísticas isoladas, mas também apresentava flexibilidade. Deste modo, decidimos expandir a proposta, sugerindo uma abordagem geral que pudesse ser aplicada para atender muitos critérios estéticos e para manipular várias classes de grafos.

Um dos objetivos deste trabalho é, portanto, descrever como heurísticas distintas podem ser combinadas em um Time Assíncrono, a fim de obterem desenhos melhores do que se executadas separadamente - característica esta que chamamos de sinergia. Estamos interessados, de um modo particular, em produzir desenhos que mostrem vários aspectos estéticos, os quais não são considerados todos por um único algoritmo de desenho, mas que podem ser satisfeitos devido à cooperação entre as heurísticas. O trabalho destaca que a abordagem é flexível e comenta outras vantagens como a facilidade de expansão e a possibilidade de se explorar paralelismo.

O restante deste trabalho está organizado como segue:

Alguns conceitos sobre Teoria de Grafos, Desenho de Grafos e sobre Times Assíncronos são introduzidos no capítulo 2.

No capítulo 3, propomos uma abordagem para desenhar grafos baseada na utilização de Times Assíncronos e sugerimos uma estrutura geral de um time que atenda a esta finalidade.

No capítulo 4, mostramos exemplos de times para desenhar grafos gerais e grafos direcionados, comentando os resultados obtidos. Discutimos também alguns aspectos de configuração dos times, relacionados com a elaboração de novos agentes, com a inclusão de critérios estéticos e com a organização de soluções na memória.

No capítulo 5, concluímos a dissertação destacando as vantagens e desvantagens do uso de Times Assíncronos para Desenho de Grafos. Propomos ainda alguns estudos deixados à posteriori.

Capítulo 2

Conceitos Gerais

Neste capítulo, introduzimos os principais conceitos sobre Desenho de Grafos e sobre Times Assíncronos empregados no decorrer da dissertação.

Apresentamos inicialmente algumas definições básicas em Teoria dos Grafos. O leitor mais acostumado com estas definições poderá avançar diretamente para a seção 2.2 sem prejuízo do entendimento do trabalho.

2.1 Grafos

As definições que se seguem baseam-se na terminologia adota em [Bol78, ST81].

Um *grafo* G (não-direcionado[1]) é um par (V, E), onde V é um conjunto finito de pontos denominados *vértices* e E é um conjunto finito de *arestas*. Uma aresta $e \in E$ é um par não ordenado xy de vértices distintos de V. A aresta e pode ser escrita como xy ou yx e indica que x e y são os extremos de e. Neste caso, dizemos que x e y são *adjacentes* e que a aresta xy é *incidente* em x e em y. O *grau* de um vértice $x \in V$ é definido como sendo o número de vértices de G adjacentes a x.

Um *caminho* de um vértice x para um vértice y em um grafo G consiste de uma seqüência $S_{xy} = (x = v_0, v_0v_1, v_1, v_1v_2, \ldots, v_{k-1}v_k, v_k = y)$ onde v_iv_{i+1} são arestas de G (para $i = 0, 1, \ldots, k-1$) e $v_0, \ldots, v_k$ são vértices de G. Sem perda de generalidade, podemos omitir os vértices da seqüência. Deste modo, um *caminho* de um vértice x para um vértice y é uma seqüência de arestas $S_{xy} = (v_0v_1, v_1v_2, \ldots, v_{k-1}v_k)$, onde $v_0 = x$, $v_k = y$, e v_iv_{i+1} são arestas de G para $i = 0, 1, \ldots, k-1$. Quando um caminho tem ambos os seus extremos iguais ($v_0 = v_k$), ele é chamado *caminho fechado* (ou *ciclo*). O *comprimento* de um caminho S_{xy} é representado por $|S_{xy}|$ e é definido como sendo o número de arestas da seqüência. A *distância* entre dois vértices x e y no grafo (também conhecida como *distância teórica*) é dada pelo comprimento do menor caminho de x para y.

Um grafo G é *conexo* se existe um caminho S_{xy} para todo par de vértices distintos x e y de G.

[1] também chamado de grafo não-orientado

O *centro teórico* de um grafo não-direcionado G é representado por um vértice x cuja soma das distâncias de x a todos os demais vértices de G é mínima.

Dizemos que um grafo $G' = (V', E')$ é um *subgrafo* de $G = (V, E)$, se $V' \subseteq V$ e $E' \subseteq E$. Se G' contém todas as arestas de G cujos extremos são vértices de V', então G' é chamado de *subgrafo induzido* de G.

Um grafo H é dito *gerado* de um grafo G, notação $H \sqsubseteq G$, se H é um subgrafo de G, e quaisquer dois vértices u e v são adjacentes em H se e somente se forem adjacentes em G.

Dois grafos são *isomorfos* se há uma correspondência entre seus conjuntos de vértices que preserva a adjacência. Assim, $G = (V, E)$ é isomorfo a $G' = (V', E')$ se existe uma função bijetora $\phi : V \rightarrow V'$, tal que $xy \in E$ implica que $\phi(x)\phi(y) \in E'$ e vice-versa. Naturalmente, grafos isormofos têm o mesmo número de vértices e de arestas.

De forma semelhante à definição de grafo não-direcionado, um *grafo direcionado* G é um par (V, E), onde V é um conjunto finito de vértices e E é um conjunto de pares **ordenados** de vértices. Num grafo direcionado, uma aresta $e = (x, y)$ de E é dita *discindente* de x e *incidente* em y, respectivamente. O vértice y é dito *adjacente* a x.

O *grau de entrada* de um vértice x de um grafo direcionado G é definido como sendo o número de arestas que incidem em x. O *grau de saída* deste vértice é dado pelo número de arestas discidentes de x. O *grau* de x é definido como a soma entre o grau de entrada e o grau de saída de x.

Um *caminho direcionado* de um vértice x para um vértice y é uma seqüência de arestas $S_{xy} = ((v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k))$, onde (v_i, v_{i+1}) são arestas orientadas de G, para $i = 0, 1 \ldots k-1$. Quando um caminho direcionado tem ambos os seus extremos iguais ($v_0 = v_k$), ele é chamado *caminho direcionado fechado* (ou *ciclo direcionado*). Um grafo direcionado é dito *acíclico* (também conhecido por *DAG*, do inglês "Directed Acyclic Graph") se não possui caminhos direcionados fechados.

Se existe um caminho direcionado de um vértice x para um vértice y em um grafo direcionado G, então dizemos que y é *alcançado* por x. Definimos o *conjunto alcançável* do vértice x, notação R_x, como sendo o conjunto de todos os vértices alcançáveis por x.

Um grafo G_s não-direcionado é *subjacente* a um grafo direcionado G se G_s é obtido a partir de G removendo-se somente a orientação de suas arestas.

Um grafo direcionado é *fracamente conexo* se seu grafo subjacente for conexo. Os grafos direcionados fracamente conexos serão referenciados, neste trabalho, apenas por "grafos conexos", suprimindo-se o termo "fracamente".

Para um grafo $G = (V, E)$, direcionado ou não-direcionado, denotamos por $|V|$ e $|E|$ os tamanhos dos conjuntos V e E, respectivamente. O *tamanho* de um grafo G, notação $|G|$, consiste da soma $|V| + |E|$.

2.2 Desenho de Grafos

Definições Básicas

Um *desenho bidimensional* de um grafo $G = (V, E)$ é uma função $D : G \rightarrow I\!R^2$ que associa a cada vértice e aresta de G um objeto em $I\!R^2$. Neste trabalho, estamos interessados somente em desenhos bidimensionais de grafos. Omitiremos o adjetivo "bidimensional", ficando este subentendido.

Intuitivamente, um algoritmo para desenhar grafos geralmente tem como entrada uma representação de um grafo numa forma combinacional, como por exemplo uma matriz ou uma lista de adjacências, produzindo como saída um desenho através da escolha de uma localização no plano para os vértices e uma dada representação para as arestas. Este processo é ilustrado na figura 2.1.

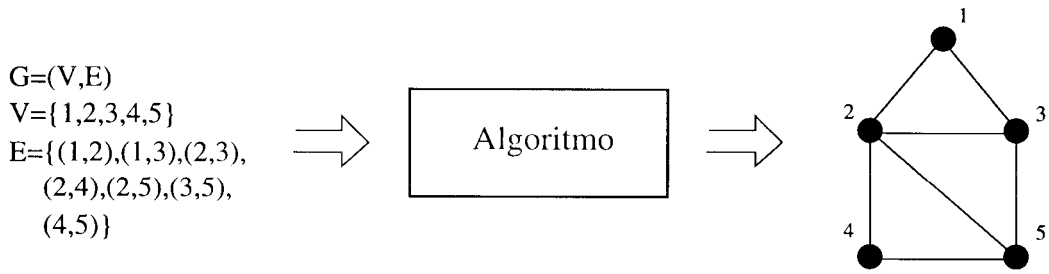


Figura 2.1: Algoritmo para desenhar grafos.

Deste modo, um algoritmo básico em Desenho de Grafos envolve três etapas:

1. escolher um padrão gráfico para vértices e arestas;
2. definir os critérios que determinam a qualidade estética do desenho;
3. e produzir um desenho segundo o padrão gráfico adotado, a fim de satisfazer os critérios estéticos definidos.

O *padrão gráfico* determina a forma com que os vértices e as arestas são representados geometricamente. Existem muitos padrões gráficos, sendo que, o mais comum é desenhar os vértices como círculos e as arestas como linhas poligonais[2]. Os pontos onde as linhas poligonais mudam de direção são chamados de *dobras* (observe a figura 2.2).

Dentro do padrão de linhas poligonais existem duas configurações de particular interesse quanto à representação das arestas: cada aresta pode ser simbolizada por um único segmento de reta, e, neste caso, o desenho é chamado de *desenho em linhas retas* (figura 2.3); ou cada aresta pode ser simbolizada por vários segmentos de retas ortogonais aos eixos do plano, e o desenho é chamado de *desenho ortogonal* (figura 2.4).

[2] Alguns autores desenham as arestas como arcos

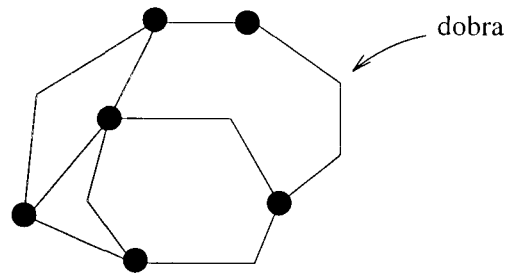


Figura 2.2: Padrão gráfico de linhas poligonais.

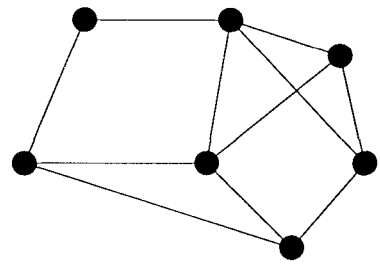


Figura 2.3: Desenho em linhas retas.

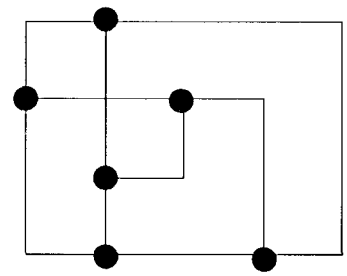


Figura 2.4: Desenho ortogonal.

Algumas terminologias têm sido empregadas para identificar certas propriedades visuais apresentadas pelos desenhos de grafos. De um modo geral, dizemos que um desenho é produzido sobre uma *grade* quando utilizamos uma região reticulada para desenhar vértices e pontos de dobra das arestas.

Um desenho de um grafo direcionado em linhas poligonais é chamado de *"upward"* quando todas as suas arestas possuem pelo menos uma componente da linha poligonal voltada para cima e nenhuma componente volta para baixo[3].

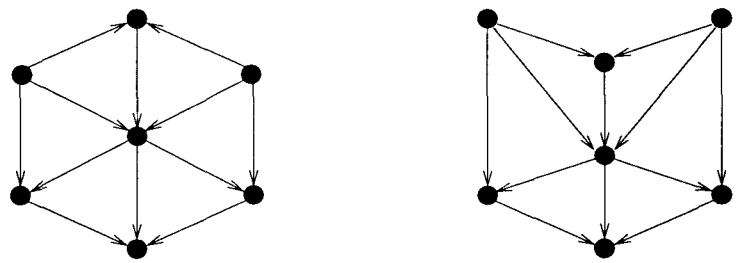
Desenhos que não apresentam cruzamentos de arestas e que não possuem sobreposição de vértices em arestas são denominados de *planares*.

É possível gerar um número infinito de desenhos diferentes para um mesmo grafo, independente do padrão gráfico escolhido. Contudo, somente uma parcela desses desenhos têm alguma utilidade prática.

Na maioria das vezes, entre todos os possíveis desenhos, estamos interessados apenas naqueles que são "legíveis", ou seja, que ajudam na visualização dos vértices e arestas, e portanto na compreensão do grafo.

Em outras situações, o importante não é obter propriamente legibilidade, mas sim, atender a certas necessidades que facilitem o mapeamento direto do desenho em algum objeto de aplicação real. Citamos como exemplo o caso do projeto de circuitos digitais, cujo objetivo maior é gerar diagramas elétricos com o mínimo possível de cruzamentos entre conexões.

A escolha de qual subconjunto de desenhos é útil, pode ser uma atividade subjetiva, que varia de acordo com o gosto particular de cada usuário. Veja, por exemplo, os desenhos de um mesmo grafo direcionado apresentados na figura 2.5. Um grande número de pessoas pode achar mais interessante o desenho 2.5(a), uma vez que este exibe simetrias; no entanto, é possível que outras pessoas gostem mais do desenho 2.5(b), pois ele mostra arestas com orientação uniforme para baixo.



(a) (b)

Figura 2.5: Subjetividade na definição do que seja um bom desenho.

Com o objetivo de eliminar essa subjetividade, definimos um conjunto de critérios que indicam quais são as características nos desenhos que nos agradam. Estes critérios são denominados, "grosso modo", de *critérios estéticos*, por estarem freqüentemente relacionados com

[3]O conceito de "upward" pode ser utilizado de forma mais intuitiva e abrangente para referenciar um desenho de um grafo direcionado em que todas as arestas estão voltadas para um único sentido, seja esse para baixo ou para cima.

características estéticas visuais. Apesar disso, eles podem refletir aspectos de outra natureza (não obrigatoriamente de caráter estético).

Alguns exemplos de critérios estéticos gerais utilizados são:

- apresentar poucos cruzamentos de arestas,
- exibir simetrias[4]
- distribuir os vértices uniformemente no espaço desenhado,
- mostrar arestas com comprimento uniforme e
- dispor as arestas numa mesma direção, tanto quanto possível (no caso de grafos direcionados).

Podemos dizer que a *qualidade* de um desenho está associada ao grau de satisfação de um conjunto de critérios estéticos. Dependendo de quanto um desenho atende a certo critério, ele pode ser considerado como de boa ou de má qualidade. A busca por desenhos de boa qualidade é comumente analisada na literatura como problemas de **Otimização Multiobjetiva.**

Além de critérios estéticos, podemos definir também "restrições" para orientar a elaboração dos desenhos. As restrições estabelecem regras que devem ser cumpridas a qualquer custo. Como exemplo: um conjunto de vértices deve ser desenhado obrigatoriamente em uma certa posição ou de um modo específico; ou determinados vértices e arestas devem ser desenhados seguindo um padrão gráfico diferente dos demais. Deste modo, uma restrição nunca deve ser desrespeitada, ao passo que um critério estético pode ser atenuado dentro de um certo grau de tolerância. No presente trabalho, buscamos satisfazer mais a critérios estéticos do que a restrições, em virtude da menor "rigidez" exigida por essa abordagem e da existência de soluções de compromissos, explicadas mais adiante.

A atividade de desenhar grafos em linhas poligonais divide-se em dois paradigmas: o posicionamento dos vértices ("placement") e o roteamento das arestas ("routing"). O primeiro paradigma consiste em adotar um padrão gráfico para desenhar as arestas e encontrar uma localização para os vértices que atenda a alguns critérios estéticos ou restrições. O segundo consiste em fixar a posição dos vértices e satisfazer a critérios estéticos ou restrições roteando as arestas do grafo.

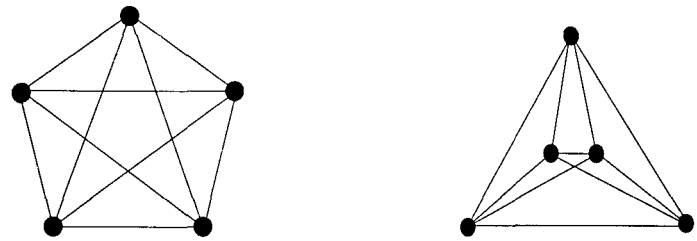
A busca por desenhos de boa qualidade, tanto para problemas de posicionamento como de roteamento, incorre em muitas dificuldades. A principal dificuldade na área de Desenho Grafos segue do fato de que a obtenção de bons desenhos freqüentemente está associada a problemas *NP-Difíceis* [DETT94, Ead89].

Devemos observar, também, que nem sempre existe um desenho de um grafo que satisfaça de modo simultâneo dois ou mais critérios estéticos. Na verdade, é comum ocorrer situações em que a satisfação ampla de um determinado critério implica no relaxamento de outro. Quando

[4] Definimos simetria informalmente, mais pela facilidade de sua visualização, como propriedades de rotação e reflexão do desenho.

ocorre esse tipo de problema, dizemos que os critérios estéticos são *conflitantes*. Nas figuras 2.6 e 2.7, ilustramos alguns exemplos de conflitos.

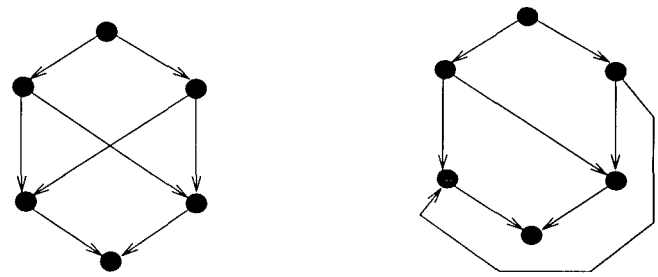
A figura 2.6 mostra dois desenhos de um grafo completo de 5 vértices. O desenho 2.6(a) possui o máximo de simetrias, contudo, apresenta 5 cruzamentos. Já o desenho 2.6(b) apresenta o mínimo de cruzamentos, entretanto exibe menos simetrias (o desenho 2.6(b) não possui a simetria de rotação vista no desenho 2.6(a)).



(a) (b)

Figura 2.6: Conflito entre os critérios de poucos cruzamentos e muitas simetrias.

A figura 2.7 mostra desenhos de um grafo direcionado acíclico. O desenho 2.7(a) é "upward", enquanto o desenho 2.7(b) é planar. Observe que não existe um desenho desse grafo que seja planar e "upward" simultaneamente.



(a) (b)

Figura 2.7: Conflito entre os critérios de orientação uniforme e poucos cruzamentos.

Muitos conflitos têm sido relatados na literatura [DH89] envolvendo critérios estéticos como: mínimo número de cruzamentos, mínima área do desenho, mínimo número de dobras das arestas, máximo número de simetrias, maior resolução angular, entre outros. A existência de conflitos torna mais difícil a atividade de desenhar grafos sarisfazendo um conjunto de critérios estéticos. Em geral, procura-se resolver os conflitos buscando por soluções de compromisso, as quais atendem de forma equilibrada os diversos critérios pré-determinados [DH89, Tun93].

Face à complexidade dos problemas que envolvem a atividade de desenhar grafos, justifica-se o uso de heurísticas na produção de boas soluções.

Um objetivo de singular importância na área de Desenho de Grafos é projetar algoritmos *flexíveis*, ou seja: algoritmos cuja inclusão de novos critérios estéticos ou cuja modificação para que trabalhem com outra classe de grafos não implique em um grade esforço de codificação.

Motivados pela vasta quantidade de heurísticas disponíveis para desenho de grafos [DETT94], propomos uma abordagem onde algoritmos são organizados de modo que cooperem, produzindo bons desenhos, e de uma forma flexível

A seguir, introduzimos algumas técnicas referentes ao desenho de quatro classes genéricas de grafos, dando enfoque ao desenho de grafos gerais e de grafos direcionados. Esta escolha deve-se ao fato dessas duas classes serem alvo de intensa pesquisa na área.

2.2.1 Árvores

Historicamente, as árvores estão entre as primeiras classes estudadas em Desenho de Grafos, em função da sua popularidade e da simplicidade de sua estrutura.

As técnicas para desenhar árvores consideram dois casos distintos: desenho de Árvores com Raiz ("Rooted Trees") e desenho de Árvores Livres ("Free Trees"). Em ambos os casos, é desejável a obtenção de desenhos planares em linhas poligonais.

Árvores com Raiz

As Árvores com Raiz representam hierarquias, tais como diagramas organizacionais, árvores de busca, árvores de chamadas de sub-rotinas, etc. Entre os critérios estéticos para desenho de Árvores com Raiz podemos encontrar:

- os vértices devem ser distribuídos sobre níveis hierárquicos horizontais, de acordo com a sua profundidade na árvore;
- para árvores binárias, os filhos esquerdo e direito de um vértice devem ser desenhados, respectivamente, à esquerda e à direita do seu pai;
- os pais devem estar centralizados sobre seus filhos;
- e a largura do desenho deve ser mínima.

Um dos primeiros trabalhos nessa área apareceu em [WS79], onde Wetherell e Shannon apresentaram um algoritmo para desenho de árvores binárias que satisfaz os três primeiros critérios mencionados.

Posteriormente, Reingold e Tilford mostraram em [RT81] que o algoritmo de Wetherell e Shannon poderia gerar desenhos muito mais largos do que o necessário e com algumas outras características estéticas indesejáveis. Eles apresentaram então um novo algoritmo e incluíram os seguintes critérios: mostrar simetrias e desenhar sub-árvores isomorfas sempre do mesmo modo.

Embora o algoritmo de Reingold e Tilford obtenha resultados melhores do que aqueles apresentados por Wetherell, ele não satisfaz obrigatoriamente o critério de mínima largura. Um trabalho mais recente de Sopwit e Reingold em [SR83] mostrou que a construção de um desenho de uma árvore binária com largura mínima, tal que os vértices pais estejam centralizados sobre seus filhos e que sub-árvores isomorfas sejam congruentes (mesmo ângulo descrito pelas arestas

incidentes nos vértices pais e nos filhos) é um problema *NP-completo* quando os vértices têm coordenadas sobre uma grade. De outra forma, considerando o desenho sobre um espaço contínuo, o problema pode ser aproximado em tempo polinomial através de técnicas de programação linear.

Árvores Livres

Um maneira bastante elegante de se desenhar Árvores Livres consiste em escolher um vértice como raiz e aplicar um dos algoritmos mencionados acima para construir um representação hierárquica em níveis. Em seguida, converte-se a representação em níveis para um desenho radial, onde o vértice raiz é colocado no centro da figura e os demais vértices são posicionados em círculos concêntricos de acordo com seus níveis [Ead92].

Alguns critérios estéticos de interesse para o desenho de Árvores Livres incluem: mostrar simetrias e apresentar arestas com comprimento uniforme. As arestas são desenhadas como linhas retas ou linhas ortogonais.

Eades em [Ead92] apresenta estudos sobre desenho de Árvores Livres, destacando não apenas algoritmos para obter desenhos radiais, como também, a utilização do método Springs [Ead84, KK89]. O método Springs é muito conhecido e consiste em modelar o grafo como um sistema dinâmico de molas no $I\!R^2$, em que os vértices são associados a massas ligadas por molas. O comprimento e a constante elástica das molas dependem da distância teórica de seus vértices correspondentes. Uma simulação das forças de atração e repulsão entre as partículas é realizada para encontrar posições de mínima energia local do sistema, as quais estão associadas a desenhos do grafo. Devido à formulação do sistema, o método Springs consegue simetrias; entretanto, é discutível se ele pode ou não evitar cruzamentos de arestas, mesmo para o caso de árvores. Uma combinação interessante entre um algoritmo para desenho radial e um algoritmo que implementa o método Springs é proposta por Eades em [Ead92]. O algoritmo resultante produz desenhos com características simétricas sem cruzamentos em todos os casos testados.

Como no desenho de Árvores com Raiz, o desenho de Árvores Livres também apresenta algumas dificuldades na satisfação de certos critérios estéticos. Em especial, construir desenhos ortogonais de Árvores Livres onde os vétices têm coordenadas sobre uma grade, de tal modo que o comprimento da maior aresta seja minimizada é um problema *NP-difícil* [BC87, Bra88, Gre89].

2.2.2 Grafos Planares

Um grafo é dito *planar* se ele admite um desenho planar.

A classe de grafos planares é certamente a mais bem estudada na área de Desenho de Grafos devido a 2 fatores: a sua grande aplicação, principalmente, no projeto de circuitos VLSI, e devido ao fato de estar associada a muitos conceitos e problemas téoricos de fundamental importância em Teoria dos Grafos.

Dentre as muitas formas gráficas de se desenhar grafos planares é bastante comum produzir desenhos em linhas retas. Segundo resultados apresentados em [Wag36, Far48, Ste51], todo grafo planar pode ser desenhado sem cruzamentos utilizando-se esse padrão.

Os algoritmos para construir desenhos de grafos planares em geral incluem os seguintes passos:

1. teste de planaridade,
2. construção de uma representação planar e
3. utilização da representação planar para produzir um desenho segundo algum padrão gráfico.

O teste de planaridade tem por objetivo verificar se o grafo a ser desenhado é de fato planar. A busca por um algoritmo que realizasse o teste de planaridade em tempo linear foi um desafio de grande interesse, até ser conseguido por Hopcroft e Tarjan [HT74].

Em geral, os algoritmos que testam a planaridade podem ser modificados para construir uma representação planar do grafo. Tal representação é uma estrutura de dados que descreve as adjacências entre as faces de um desenho planar, e pode ser utilizada para obter um desenho final sem cruzamentos de arestas. Deste modo, um desenho de um grafo planar pode ser gerado em tempo linear modificando-se certos algoritmos de teste de planaridade.

Algumas convenções gráficas para o desenho de grafos planares foram introduzidas pela área de projetos de circuitos VLSI. Por exemplo, tornou-se comum desenhar os grafos planares sobre uma grade de baixa resolução, adotando-se um padrão gráfico ortogonal. Neste caso, alguns critérios estéticos desejados incluem: minimizar o número de dobras das arestas e a área ocupada pelo desenho. Entretanto, o problema de minimizar dobras é *NP-difícil* [GT95a].

Uma outra convenção motivada pelo projeto de circuitos VLSI é o conceito de *representação de visibilidade*, que consiste em representar os vértices de um grafo por segmentos horizontais e as arestas por segmentos verticais que interceptam apenas os seus vértices extremos.

Para os desenhos em linha reta, os critérios estéticos mais utilizados incluem: minimizar o comprimento das arestas e maximizar a resolução angular. A resolução angular de um desenho em linha reta consiste no valor do ângulo mínimo formado por duas arestas incidentes em um mesmo vértice. O problema da resolução angular para desenhos em linha reta é *NP-difícil* [Gar95]. Outro problema relacionado com o padrão gráfico de linha reta consiste em obter um desenho planar fixando-se previamente o comprimento das arestas, o qual também é *NP-dicífil* [EW90].

Um dos problemas em aberto para desenho de grafos planares diz respeito ao desenvolvimento de algoritmos simples para teste de planaridade em tempo linear. Todos os algoritmos de teste de planaridade existentes são difíceis de serem entendidos e exigem muito esforço de implementação, o que limita o seu uso em sistemas práticos.

Apesar da dificuldade de se implementar o teste de planaridade, algumas técnicas para desenhar grafos baseam-se no desenho de grafos planares, em função da existência de algoritmos que executam em tempo linear. A idéia principal é converter um grafo não planar em um grafo planar, através de um processo chamado de *planarização*, e produzir um desenho utilizando-se um dos algoritmos conhecidos para grafos planares. A planarização pode ser realizada através

de algumas operações como a eliminação de vértices, eliminação de arestas ou separação de vértices. Em especial, a eliminação de arestas procura remover um conjunto de arestas de um grafo para torná-lo planar. Uma vez que há interesse na obtenção do menor conjunto de arestas que devem ser eliminadas a fim de conseguir planaridade, este método de planarização torna-se equivalente ao problema de encontrar o *subgrafo planar máximo*, que é conhecido por ser *NP-difícil* [GJ79]. Existem, no entanto, algoritmos de tempo polinomial para computar um subgrafo planar maximal.

2.2.3 Grafos Direcionados

Muitas estruturas hierárquicas como, por exemplo, as redes PERT, os diagramas de chamada de subrotinas, os organogramas e os automatos finitos são representados por grafos direcionados, em particular, por grafos direcionados acíclicos (DAG's).

A técnica mais comum para a construção de um desenho de um grafo direcionado consiste em dispor os seus vértices em níveis de hierarquia, de forma semelhante ao desenho de árvores com raiz. Os níveis são simbolizados por retas horizontais dispostas uma abaixo da outra, com espaçamento uniforme, e são numerados em ordem crescente partindo da reta mais baixa (nível l_0) até a mais alta (nível l_h). Os vértices são associados aos níveis, de tal modo que as arestas fiquem voltadas para baixo, constituindo um desenho "upward" (na verdade "downward").

Obviamente uma representação "upward" só é obtida quando o grafo não possui ciclos. Neste caso, uma ordenação topológica pode ser realizada para estabelecer o nível de cada vértice.

Já em desenhos de grafos direcionados com ciclos, algumas arestas ficarão inevitavelmente voltadas para cima (ditas arestas "contrárias"), sendo importante minimizar esse efeito.

Em geral, procura-se construir uma hierarquia em que todas as arestas conectam vértices em dois níveis consecutivos. Quando uma arestas é muito longa e "corta" alguns níveis, ela é substituída por arestas menores, através da inclusão de *vértices falsos* nos pontos de interseção com os níveis. As novas arestas são representadas por linhas retas que conectam vértices em níveis consecutivos e têm a mesma orientação da aresta original. Além disso, os vértices falsos não são exibidos no desenho.

A utilização de vértices falsos permite obter desenhos com dobras nas arestas. Uma dobra ocorre quando as arestas que ligam um vértice falso a seus adjacentes nos níveis imediatamente acima e abaixo não estão dispostas formando em linha reta.

Um dos trabalhos pioneiros no desenho de grafos direcionados e que fez uso de níveis foi o de Sugiyama et al. em [STT81], onde é apresentado um método para construir uma representação hierárquica, que envolve 3 passos: no 1º passo, os ciclos são removidos e o grafo resultante (acíclico) é hierarquizado, associando-se níveis a cada vértice; no 2º passo, permuta-se a ordem dos vértices sobre os níveis, com o objetivo de reduzir o número de cruzamentos de arestas; no 3º passo, encontra-se uma posição definitiva para os vértices em cada nível, de forma que os eles fiquem melhor distribuídos ou que a distância de roteamento das arestas seja minimizada (procura-se reduzir o número de dobras das arestas ou torná-las verticais), sem alterar a ordem pré-estabelecida no passo anterior. Finalmente, os ciclos dos grafo são restaurados e um desenho

final é produzido.

Em cada passo do algoritmos de Sugiyama, tenta-se satisfazer um ou mais critérios estéticos, que são *NP-difícies* em sua grande maioria.

No primeiro passo, por exemplo, a remoção de ciclos visa obter uma orientação uniforme das arestas. Na abordagem original de Sugiyama [STT81], essa remoção é feita condensando-se ciclos em vértices. No entanto, a técnica mais utilizada é inverter temporariamente a orientação de algumas arestas do grafo de modo que os ciclos sejam desfeitos. Nesta técnica, procura-se inverter o menor número possível de arestas, pois são exatamente tais arestas invertidas que ficarão voltadas para cima no desenho final. Esse problema é conhecido na literatura por *Feedback Arc Set Problem* e é *NP-difícil* [GJ79].

No segundo passo, o problema de minimizar cruzamentos também é *NP-difícil*, mesmo quando considerada uma hierarquia com apenas dois níveis e mantendo-se fixa a ordem dos vértices em um dos níveis [EW]. Sugiyama apresentou uma heurística para reduzir cruzamentos, denominada de *Baricentro*, e que tem sido bastante utilizada em muitos algoritmos para desenho de grafos direcionados. Essa heurística é explicada abaixo dada a sua importância.

O Baricentro consiste em colocar os vértices de um nível em sua posição média (ou baricêntrica) em relação a seus adjacentes num nível consecutivo. Um algoritmo que implementa o Baricentro geralmente executa como segue: a posição dos vértices do nível mais alto l_h é fixada e o Baricentro é aplicado para os vértices do próximo nível, l_{h-1}, tomando como base os seus adjancentes em l_h. Esse processo continua para os níveis abaixo, até que os vértices no nível l_0 sejam colocados em sua posição baricêntrica. Em seguida, o mesmo processo é repetido de baixo para cima, iniciando no nível l_0. O Baricentro é aplicado sucessivas vezes para cima e para baixo até que não se consiga mais reduzir o número de cruzamentos de arestas.

Uma heurística semelhante ao Baricentro é conhecida como *Mediana* e executa colocando vértices na posição média entre os seus adjacentes mais à esquerda e mais à direita .

Um outro critério estético importante, satisfeito em parte nos passos 1 e 2, consiste em distribuir os vértices uniformente sobre uma região, gerando desenhos que não sejam nem muito largos nem muito compridos. O problema de encontrar uma disposição hierárquica que satisfaça de forma ótima esse critério é *NP-difícil* [Ead89].

2.2.4 Grafos Gerais

Tratamos, nesta seção, de grafos não-direcionados gerais, que abreviamos apenas por *grafos gerais*.

Esta categoria é composta dos grafos para os quais não se dispõe de muita informação sobre sua estrutura, além do conhecimento de seus vértices e arestas. Construir desenhos para essa classe de grafos parece ser um tarefa desafiadora, visto que não existe nenhuma regra bem definida que indique onde posicionar os vértices. Tal situação é bem diferente daquela que caracteriza os grafos direcionados e as árvores, para os quais sabemos que as arestas devem ficar voltadas para baixo.

Apesar dessa aparente dificuldade, alguns critérios estéticos bastante intuitivos têm sido

propostos para desenhar grafos gerais como, por exemplo: mostrar muitas simetrias, distribuir os vértices uniforme e apresentar poucos cruzamentos. Em geral, a satisfação destes critérios implica em desenhos que ajudam na compreensão do grafo. Devemos observar, contudo, que alguns dos critérios estão relacionados com problemas *NP-difíceis*, tais como: identificar todas as simetrias apresentadas por um desenho [Man91, Man90] e minimizar cruzamentos [GJ83].

Uma possível abordagem para desenhar grafos gerais consiste em transformá-los em uma outra classe de grafos e aplicar os algoritmos adequados a esta classe. Citamos como exemplo: planarização e orientação [DETT94]. O problema desta abordagem é que dificilmente são obtidos desenhos que possuam caractéristicas estéticas além daquelas implícitas nos algoritmos utilizados. Um outro problema é que os desenhos podem apresentar características não desejadas, comuns à classe para a qual o grafo foi convertido.

Uma abordagem bastante promissora, que obtém bons desenhos, consiste em modelar o grafo como um sistema físico, onde os vértices são partículas sobre as quais atuam forças de atração e repulsão. Uma disposição inicial para cada vértice é escolhida e o sistema é simulado de modo a alcançar uma configuração final de estabilidade, associada a um desenho do grafo. O método Springs [Ead84, KK88] é um exemplo dessa abordagem. Um outro método, de caráter mais geral, que adota a modelagem de um sistema físico é a proposta de Davidson e Harel [DH89] baseada em Simulated Annealing. A seguir apresentamos esses dois métodos:

Springs

A primeira abordagem de um sistema energético como descrito no parágrafo anterior foi o método Springs proposto por Eades em [Ead84], e que foi aperfeiçoado posteriormente por Kamada e Kawai em [KK88, KK89]. No método Springs, um grafo é modelado como um sistema dinâmico de molas no plano, em que cada vértice é representado por uma partícula e cada par de vértices está conectado por uma mola. O comprimento e a constante elástica da mola dependem da distância teórica entre os vértices correspondentes no grafo. O objetivo é encontrar um estado de baixa energia do sistema, que esteja associado a um bom desenho. O método trabalha basicamente como segue: dado uma configuração inicial aleatória, o vértice mais "crítico" (que contribui com a maior parcela de energia) é movido para uma posição que minimiza a energia total do sistema; os demais vértices são mantidos "congelados" na sua posição original. Esse passo é repetido sucessivamente para o próximo vértice mais crítico, até que não haja mais vértices que contribuam com valores "altos" à função de energia. As posições para os vértices que minimizam a energia do sistema são encontradas através da aplicação do método de Newton-Raphson.

O Springs é conhecido por gerar desenhos 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 [Lin92].

Proposta de Davidson e Harel Baseada em Simulated Annealing

Simulated Annealing (SA) é um método de otimização iterativo, formulado inicialmente por

Kirkpatrick *et al* [KJV83] e que se originou na Mecânica Estatística [MRR+53]. O método tenta escapar de mínimos locais utilizando operações análogas ao processo em que líquidos são esfriados até assumirem uma forma cristalina, processo esse chamado de *annealing*.

A motivação do método segue da observação dos princípios que regulam o processo *annealing*: se o esfriamento de um líquido for lento o suficiente, então as pequenas forças iônicas de cada átomo interagem e organizam a estrutura do líquido, atingindo uma forma chamada de cristal, que está associada ao estado de mínima energia do sistema. Entretanto, se este esfriamento for demasiadamente rápido, o sistema (líquido) assume uma forma amorfa que representa mínimos locais.

Durante o *annealing*, o sistema obedece a distribuição de Boltzmann:

$$P(E) \simeq e^{-\frac{E}{kT}},$$

que define uma distribuição de probabilidades para estados de energia E, como uma função da temperatura T e da constante de Boltzmann k.

Metropolis e outros em [MRR+53] sugeriram um procedimento para simular esse processo em uma dada temperatura através de uma seqüência de movimentos que mudam o estado do sistema. Novos estados são obtidos dentro da "vizinhança" do estado corrente, isto é, pertubando-se uma pequena parcela do sistema. A regra básica do procedimento é que a probabilidade de passar de um estado com energia E_1 para um novo estado com energia E_2 é dada pela função:

$$p = min\{1, e^{-\frac{E2-E1}{kT}}\}.$$

Isto significa que o sistema sempre muda para o novo estado se $E_2 < E_1$. Caso contrário, se $E_2 > E_1$, o novo estado será aceito com probabilidade p.

Um algoritmo para simular o *annealing*, proposto por Kirkpatrick e outros em [KJV83], consiste em estabelecer um alto valor para a temperatura T e ir diminuindo-o lentamente, enquanto se atinge um estado de equilíbrio através da aplicação sucessiva de pequenas pertubações no sistema para cada valor de T. Os passos básicos de um algoritmo para Simulated Annealing é apresentado na figura 2.8. Kirkpatrick e outros mostraram que este processo pode ser aplicado para resolver problemas gerais de otimização.

Desta forma, Davidson e Harel em [DH89] fizeram uso de Simulated Annealing para desenhar grafos gerais. Eles definiram uma função energética de custo associada aos estados do sistema, que representa alguns critérios estéticos como: distribuição uniforme dos vértices sobre uma região de desenho, comprimento mínimo das arestas, mínimo número de cruzamentos, distância máxima entre vértices e arestas. A função energética total de um estado σ referente a um grafo $G = (V, A)$ tem a seguinte forma:

$$E(\sigma) = \lambda_1 \sum_{\forall i,j \in V} 1/{d_{ij}}^2 + \lambda_2 \sum_{\forall i \in V, k=1,2,3,4} 1/{d_{iB_k}}^2 + \lambda_3 \sum_{\forall r \in A} {l_r}^2 + \lambda_4 X + \lambda_5 \sum_{\forall i \in V, \forall r \in A} 1/d_{ir}$$

onde: d_{ij} é a distância entre os vértices i e j, d_{iB_k} é a distância entre o vértice i e a borda k da região retangular do desenho, l_r é o comprimento da aresta r, X é o número total de

1. Escolha uma configuração inicial σ e uma temperatura inicial T;
2. Repita (comumente um número fixo de vezes)
   (a) Escolha uma nova configuração σ' a partir da vizinhança de σ;
   (b) Sejam E e E' os valores da função de custo (energia) de σ e σ', respectivamente; se $Random < e^{(E-E')/T}$ então faça $\sigma \leftarrow \sigma'$;
3. Decremente a temperatura T;
4. Se uma condição de parada for satisfeita, então pare. Caso contrário, volte ao passo 2.

Figura 2.8: Algoritmo para Simulated Annealing.

cruzamentos, d_{ir} é a distância entre o vértice i e a aresta r (com r não incidente a i), e λ_1, λ_2, ..., λ_5 são constantes que ponderam as componentes da função energética. O peso de cada componente de energia determina a qualidade estética dos desenhos finais produzidos.

Essa proposta, utilizando Simulated Annealing, é considerada uma das abordagens existentes mais flexíveis para desenhar grafos, uma vez que um novo critério estético pode ser considerado através da sua inclusão na função de energia.

Em contra-partida, o Simulated Annealing é reconhecidamente lento [AK89] e exige inúmeros testes e ajustes delicados de configuração de seus parâmetros até que o sistema seja capaz de produzir bons desenhos. Entre os parâmetros que precisam ser configurados, citamos: a escolha dos pesos para os critérios, a taxa de decréscimo da temperatura e a definição da vizinhança entre os estados do sistema.

Outras Abordagens

Tunkenlang [Tun93] apresenta uma abordagem prática para desenho de grafos gerais que faz uso de uma função energética semelhante àquela do Simulated Annealing de Davidson e Harel. A novidade nesta abordagem é que uma série de otimizações são implementadas a fim de agilizar a atividade de desenho. A principal otimização consiste em gerar o desenho de forma incremental, adicionando-se um vértice por vez à figura parcialmente construída. A cada inclusão, realiza-se uma seqüência de passos iterativos para minimizar a energia do sistema, reorganizando a posição dos vértices. A proposta de Tunkelang caracteriza-se por "combinar" técnicas de otimização, permitindo construir desenhos rapidamente.

Uma outra abordagem para desenhar grafos gerais faz uso de Algoritmos Genéticos [BBS96, KA96]. Os Algoritmos Géneticos [Gol89] são processos de busca baseados nas leis da seleção natural e da genética, e consistem de três operações básicas: seleção de uma subpopulação de cromossomos a partir de uma população inicial (os cromossomos simbolizam soluções do problema); aplicação de operações genéticas de permutação e mutação sobre os cromossomos selecionados com o objetivo de gerar um conjunto de cromossomos filhos; e substituição dos

cromossomos pais pelos filhos com base em alguma regra de aptidão.

O uso de Algoritmos Genéticos para desenho de grafos sugere algumas vantagens como a possibilidade de se obter uma "população" de desenhos diferentes e o potencial de se explorar paralelismo. Entretanto, tem sido extremamente difícil codificar heurísticas inteligentes nesta abordagem, como também, ajustar os algoritmos visando a convergência para boas soluções [BBS96, KA96].

2.2.5 Resumo da Complexidade dos Problemas

Na tabela 2.1, relacionamos a complexidade de tempo de alguns dos principais problemas em Desenho de Grafos.

Observe que problemas aparentemente semelhantes podem ter complexidades diferentes. Por exemplo, tanto o teste para verificar se o grafo é acíclico (e, portanto, se permite um desenho "upward") quanto o teste de planaridade podem ser feitos em tempo linear; no entanto, o teste de planaridade "upward" é *NP-difícil*. Planaridade e estrutura acíclica são condições necessárias mas não suficientes para um desenho planar "upward".

Lembramos que alguns dos critérios estéticos mencionados são dois a dois conflitantes.

Classe de Grafo	**Problema**	**Complexidade**		**Referências**
árvore com raiz	minimizar área de um desenho planar "upward" de linhas retas sobre uma grade que mostre simetrias e isomorfismos de subárvores		*NP-difícil*	[SR83]
árvore livre	minimizar o comprimento total/máximo de arestas em um desenho ortogonal sobre um grade (o grau máximo dos vértices é 4)		*NP-difícil*	[BC87, Bra88, Gre89]
grafo planar	obter um desenho ortogonal sobre uma grade com mínimo número de dobras de arestas (o grau máximo dos vértices é 4)		*NP-difícil*	[GT95a]
grafo planar	gerar um desenho planar de linhas retas com resolução angular máxima		*NP-difícil*	[Gar95]
grafo planar	obter um desenho planar de linhas retas com o comprimento das arestas prefixado		*NP-difícil*	[EW90]
grafo em 2 níveis	minimizar cruzamentos de um desenho em nível, com a ordem dos vértices de um dos níveis predefinida		*NP-difícil*	[EW]
grafo direcionado	teste de planaridade "upward"		*NP-difícil*	[GT95b]
grafo direcionado	minimizar altura e largura de um desenho em níveis de hierarquia, distribuindo uniformemente os vértices do grafo		*NP-difícil*	[Ead89]
grafo geral	computar o número máximo de simetrias em um desenho		*NP-difícil*	[Man91, Man90]
grafo geral	minimizar cruzamentos		*NP-difícil*	[GJ83]
grafo geral	computar um subgrafo planar máximo		*NP-difícil*	[GJ79]
grafo geral	teste de planaridade	$O(\|V\|)$	$\Omega(\|V\|)$	[BL76, ET76, HT74, LEC67]
grafo geral	computar um subgrafo planar maximal	$O(\|V\| + \|E\|)$	$\Omega(\|V\| + \|E\|)$	[DT89, HT93, La 94, Dji95]

Tabela 2.1: Complexidade de alguns problemas em Desenho de Grafos.

2.3 Times Assíncronos

2.3.1 Definições e Características

Os *Times Assíncronos* (ou "A-Teams", do inglês "Asynchronous Teams") [dST91, TdS92, dST93, dS93] têm sido utilizados com uma metodologia para tratar Problemas de Otimização Combinatória. Eles consistem de uma organização de agentes autônomos que se comunicam assincronamente através de memórias compartilhadas e que formam fluxos cíclicos de dados. Cada agente de um Time Assíncrono pode implementar um algoritmo diferente para o problema em estudo. No entanto, tais agentes formam um "time" quando integrados na organização, cooperando entre si para obterem resultados melhores do que quando executados individualmente.

Neste trabalho, utilizamos a expressão "time" para indicar um Time Assíncrono específico subentendido no texto.

As principais propriedades que caracterizam os Times Assíncronos são:

Autonomia dos agentes - não há um agente supervisor que controla ou influência a execução dos outros agentes. Cada agente executa do modo independente e contribui com o seu trabalho para atingir um objetivo comum. Além disso, novos agentes podem ser adicionados à organização e agentes antigos podem ser removidos sem necessidade de notificar os demais.

Comunicação assíncrona - os agentes se comunicam sem haver sincronismo, trocando informações através de memórias compartilhadas. Os resultados gerados por um agente são armazenados nas memórias e podem ser recuperados posteriormente.

Fluxo cíclico de dados - os agentes acessam as memórias continua e iterativamente para ler e escrever informações. Nesse processo, eles formam um fluxo cíclico de dados que permitem a auto-realimentação do time.

Em geral, as memórias dos Times Assíncronos armazenam soluções (parciais ou completas) de algum problema.

Um agente chamado de *iniciador* é encarregado de preencher as memórias com soluções iniciais para serem melhoradas pelo time.

A maioria dos outros agentes trabalha produzindo uma nova solução com base nas soluções obtidas da memória.

Uma vez que as memórias apresentam limitações em termos de tamanho, algumas soluções precisam ser eliminadas para dar lugar às novas soluções. Esta tarefa é feita por um agente especial denominado *destruidor*.

Um exemplo de um Time Assíncrono genérico pode ser visto na figura 2.9, onde ilustramos uma organização composta de 2 memórias e 6 agentes. As memórias do time são representadas por retângulos. Os agentes são identificados por arcos saindo e entrando nas memórias, sendo que **I** é um agente iniciador, **D** é um agente destruidor de soluções e **A**, **B**, **E** e **F** são agentes

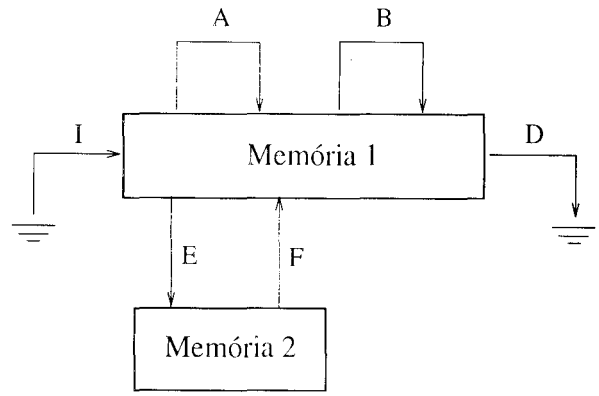


Figura 2.9: Representação gráfica de um Time Assíncrono

que compõem novas soluções. Esta representação gráfica é sugerida por Talukdar e Souza em [TdS92, dST93].

Observe a existência de fluxos cíclicos de dados no diagrama. Os agentes **A** e **B** produzem fluxos independentes, lendo e escrevendo soluções na memória **1**; e os agentes **E** e **F**, em conjunto, originam um outro fluxo que interliga as memórias **1** e **2**.

Nas próximas seções, descrevemos mais detalhadamente as características das memórias e dos agentes que compõem os Times Assíncronos.

Memórias

Duas funções são atribuídas às memórias dos times: armazenar dados e servir como meio de comunicação entre os agentes.

As memórias podem armazenar diversos tipos de dados, tais como: soluções completas e refinadas, soluções parciais (que servem para gerar soluções completas) e a descrição do problema. Qualquer outra informação útil para a execução do time também pode ser colocada na memória.

A memória que armazena soluções completas e refinadas é geralmente considerada a memória principal do time. Após o processamento, ela contém as melhores soluções que foram produzidas pela organização de agentes. As outras memórias, em sua grande maioria, atuam apenas como "buffers", armazenando temporariamente algumas soluções. Uma das características dos "buffers" é que eles não precisam ser iniciados ou terem as suas soluções removidas por agentes destruidores, visto que estas atividades são executadas pelos próprios agentes que lêem ou escrevem soluções.

O tamanho de uma memória é dado pelo número máximo de soluções que ela comporta. Deve ser observado que o tamanho da memória principal é um parâmetro que influencia no desempenho do Time Assíncrono, uma vez que pode reduzir a diversidade de soluções.

A diversidade é um fator importante pois indica que há soluções diferentes na memória e que, conseqüentemente, existe alguma chance de se produzir bons resultados. Quando o time apresenta pouca diversidade, ele tende a convergir para mínimos locais próximos às primeiras soluções obtidas, e dificilmente consegue encontrar soluções melhores.

Foi sugerido por Souza em [dS93], com base em resultados experimentais, que o tamanho das memórias deve ser proporcional ao tamanho da instância do problema para se atingir "razoável" diversidade. Memórias pequenas resultam em menos diversidade, ao passo que memórias grandes não interferem na qualidade dos resultados obtidos, embora demandem maior tempo de computação para produzirem as mesmas soluções. O tamanho ideal das memórias deve ser linearmente proporcional ao tamanho da instância do problema, quando se trata de problemas de um único objetivo. No caso de problemas multiobjetivos, o time apresenta algumas diferenças, as quais são tratadas na seção 2.3.2. Outro resultado que convém mencionar é que o tamanho das memórias que funcionam como "buffers", em geral, não interferem no desempenho do time.

Para que muitos dos agentes de um Time Assíncrono possam trabalhar, é necessário que a memória principal seja iniciada com soluções. Este processo pode ser feito preenchendo a memória com soluções produzidas de forma aleatória ou com soluções geradas por algoritmos heurísticos disponíveis para o problema. A estratégia de iniciação aleatória proporciona uma maior diversidade de soluções, enquando que o outro método permite uma convergência mais rápida, ao custo de menos diversidade. É possível, ainda, efetuar um preechimento misto, em que parte das soluções é gerada aleatoriamente e parte é gerada por algoritmos heurísticos.

As soluções produzidas pelos agentes são avaliadas segundo algum critério qualitativo e dispostas na memória em ordem de qualidade. Após os dados na memória estarem ordenados, torna-se fácil obter uma informação específica como, por exemplo, a solução de melhor ou de pior qualidade.

Agentes

Os agentes são os elementos ativos dos Times Assíncronos, responsáveis por realizar alguma tarefa. A estrutura de um agente consiste basicamente de três partes: uma **interface para leitura** de informações das memórias, um **algoritmo** para manipulação destas informações e uma **interface para escrita** das informações produzidas.

A recuperação de uma informação é feita pela interface de leitura, geralmente, obtendo uma cópia dos dados armazenados nas memórias. A escolha da informação a ser recuperada é direcionada por uma *política de seleção*. Constituem-se exemplos de políticas de seleção, algumas regras como: "ler a solução de melhor qualidade", "ler a solução de pior qualidade" ou "ler uma solução qualquer".

Os algoritmos implementados pelos agentes podem ser métodos heurísticos, algoritmos exatos para resolver uma parte menor do problema, ou algoritmos completamente aleatórios. Um agente pode, inclusive, implementar internamente um outro Time Assíncrono, sendo que esse detalhe não precisa ser mencionado na especifição do time principal. De certo modo, o código do algoritmo deve ficar encapsulado no agente por uma interface bem definida, sendo de maior importância descrever apenas a sua funcionalidade.

Os agentes do time comumente são classificados de acordo com a forma como manipulam as soluções armazenadas nas memórias. Entre os principais tipos de agentes, destacamos:

- **Agentes de Melhoria** - Através da modificação de uma solução inicial, estes agentes

produzem uma nova solução que pode ser qualitativamente melhor;

- **Agentes de Consenso** - produzem uma nova solução utilizando o que duas ou mais soluções têm em comum.
- **Agentes de Construção** - constroem uma nova solução partindo da especificação do problema ou de uma solução incompleta;
- **Agentes de "Deconstrução"** - produzem soluções parciais removendo informações de soluções completas. As soluções parciais podem ser utilizadas para compor novas soluções completas pelos algoritmos de Construção.

Outras categorias de agentes também podem ser identificadas e algumas das categorias que apresentamos podem ser divididas em sub-grupos de acordo com o problema tratado.

Além de agentes para produzir novas soluções, os Times Assíncronos também possuem agentes para eliminar soluções antigas – os agentes destruidores. A tarefa de destruir soluções é explicada a seguir.

Destruição de Soluções

A destruição de soluções é uma atividade de fundamental importância, não apenas porque abre espaço para que novas soluções sejam inseridas nas memórias, mas também porque favorece o processo de convergência dos Times Assíncronos.

Através de *Políticas de Destruição*, os agentes destruidores escolhem as soluções que serão removidas das memórias. Essas políticas estão relacionadas com a qualidade das soluções, sendo comum eliminar soluções consideradas "pouco promissoras".

Entre as políticas de destruição mais conhecidas citamos:

- **destruição da pior solução** - o agente destruidor sempre elimina a solução de pior qualidade;
- **destruição de qualquer solução com distribuição uniforme de probabilidade** - todas as soluções têm mesma chance de serem eliminadas, exceto a de melhor qualidade, que tem probabilidade zero;
- **e destruição de soluções com distribuição linear de probabilidade** - o agente segue uma distribuição de probabilidade para eliminar soluções, que cresce linearmente da melhor para a pior solução; a solução de melhor qualidade na memória tem probabilidade zero.

A política que escolhe a pior solução proporciona uma convergência rápida, mas pode ocasionar perda da diversidade. Já a política que permite a escolha de uma solução qualquer garante diversidade, embora aumente o tempo exigido para a convergência. Um compromisso entre a diversidade e o tempo de convergência é conseguido adotando-se a política de destruição linear

de probabilidades. Esses resultados são confirmados por Souza em [dS93] e por Peixoto em [Pei95].

Nas próximas seções, apresentamos algumas propriedades associadas com a vantagem de se utilizar Times Assíncronos. Em particular, comentamos sobre a capacidade de se obter sinergia, sobre a eficiência em escala da organização de agentes e sobre o potencial para explorar paralelismo e distribuição.

Sinergia

Nos Times Assíncronos, cada agente pode escrever seus melhores resultados nas memórias e os outros agentes podem ler esses dados e utilizá-los para produzir novas soluções. Uma vez que isto é possível, há chances de que um agente melhore ainda mais as soluções geradas pelos demais agentes, o que constitui uma forma de cooperação entre eles.

Se os agentes cooperam e conseguem produzir soluções melhores do que quando executados isoladamente, então dizemos que há *sinergia* na organização. A sinergia é conseqüência do esforço conjunto dos agentes e implica que o resultado produzido pelo time como um todo é maior do que a soma dos resultados gerados por cada agente.

Portanto, uma das grandes vantanges de se utilizar Times Assíncronos é obter sinergia.

Eficiência em Escala

Uma outra característica dos Times Assíncronos é que eles são eficientes em escala. Segundo Talukdar e Souza [TdS92, TdS90] uma organização é *eficiente em escala* se ela é aberta (cresce facilmente) e se é efetiva no sentido de que seus membros cooperam conseguindo resultados melhores com a inclusão de novos agentes.

Para facilitar a compreenção dos conceitos de eficiência em escala e de organização aberta e efetiva, suponha que p seja um atributo de performance como, por exemplo, a qualidade dos resultados obtidos ou uma medida de velocidade, e que A seja um conjunto de agentes. Dizemos que uma organização é *aberta* para A se qualquer agente de A pode ser incluído nesta organização. Além disso, uma organização é *efetiva* sobre A e p se ela faz com que os elementos de A cooperem entre si, melhorando os valores de p. Mais especificamente, existe pelo menos uma ordem para se incluir os agentes na organização que produza uma melhoria monotônica em p. Falamos assim que uma organização é eficiente em escala sobre A e p se ela é aberta para A e efetiva sobre A e p.

Note que a eficiência em escala possui um ponto de saturação, o que significa que a performance não melhora indefinidamente com a inclusão de novos agentes.

Paralelismo e Distribuição

Os Times Assíncronos apresentam um alto grau de paralelismo e são adequados ao processamento distribuído, como demonstrado por Souza em [dS93]. A autonomia dos agentes e a comunicação assíncrona permitem executar os agentes em uma máquina paralela ou distribuí-los sobre uma

rede de computadores, tendo-se o cuidado de manter a integridade das informações armazenadas nas memórias.

Testes com paralelismo de Times Assíncronos são apresentados por Souza em [dS93] e por Peixoto em [Pei95], e mostram um *speed up* linear no tempo de resposta para se conseguir a melhor solução de certos problemas, em função do número de processadores utilizados. Paralelismo e distribuição podem ser explorados nos Times Assíncronos implementando-se os agentes como "threads" ou como processos independentes, entre outros meios.

2.3.2 Aplicação de Times Assíncronos

De um modo geral, os Times Assíncronos são adequados para a resolução de uma classe de problemas denominada por Souza em [dS93] de *Problemas Multi-Algorítmicos* (PMA). Essa classe envolve problemas para os quais: (1) não se conhece algoritmos que os resolvam de forma exata e em tempo polinomial; (2) existem algoritmos que conseguem obter uma solução aproximada ou uma solução infactível fácil de ser ajustada a soluções factíveis, utilizando métodos heurísticos ou de relaxação do problema; e (3) os algoritmos existentes podem ser utilizados de modo iterativo a fim de melhorar continuamente as soluções obtidas.

Deste modo, muitos problemas de Otimização, principalmente os de Otimização Combinatória, estão na classe PMA e permitem o uso de Times Assíncronos.

Entre os trabalhos bem sucedidos com Times Assíncronos, destacamos o seu emprego para resolver *problemas TSP* [dS93], para o *projeto de manipuladores de robôs* [Mur92], para o *diagnóstico de falhas em sistemas de transmissão de energia elétrica* [Che93], para o *"Flow Shop Problem"* [Pei95] e para o *"Job Shop Scheduling Problem"* [Cav95].

Uma metodologia de especificação de Times Assíncronos para problemas de Otimização Combinatória foi apresentada por Peixoto em [Pei95], no intuito de ajudar na construção de um A-Team. A metodologia prevê alguns passos como: especificação do problema; definição da representação e padrão de qualidade das soluções; adequabilidade dos problemas ao uso de Times Assíncronos; classificação dos algoritmos disponíveis; elaboração das estrutura do time, etc.

Posteriormente, Rodrigues em [Rod96, RdS95] estendeu os trabalhos de Souza [dS93] e de Peixoto [Pei95], descrevendo como Times Assíncronos podem ser aplicados para tratar problemas de Otimização Multiobjetiva. Alguns conceitos apresentados por Rodrigues são descritos na próxima seção, e são utilizados no capítulo 3 na abordagem para Desenho de Grafos baseada no uso de Times Assíncronos.

Times Assíncronos para Problemas Multiobjetivos

Um Problema de Otimização Multiobjetiva pode ser definido do seguinte modo [HPYM80]:

$$min f(x) = \{f_1(x), f_2(x), \ldots, f_k(x)\}$$
$$\text{s. a. } g_i(x) \leq 0,\ i = 1, 2, \ldots, r$$

onde x é um vetor s-dimencional de variáveis de decisão, as funções g_i $(i = 1, 2, \ldots, r)$ são restrições do problema e $f_1, f_2, \ldots, f_k$ são funções objetivo.

Para tais problemas, os objetivos podem ser conflitantes entre si, não existindo uma solução ótima. A busca pela solução ótima é portanto substituída pela tentativa de se encontrar soluções de compromisso, que satisfaçam de forma equilibrada os objetivos do problema.

O uso Times Assíncronos foi proposto por Rodrigues em [Rod96, RdS95] como um método para resolução de Problemas Multiobjetivos. Entre as vantagens oferecidas por esse método destacam-se:

- geração de múltiplas soluções diferentes;
- possibilidade de obtenção de soluções de compromisso entre todos os objetivos envolvidos;
- combinação dos objetivos em uma única função de modo a priorizar um ou mais deles, caso seja desejável;
- utilização de múltiplos algoritmos, sendo que cada algoritmo satisfaz um objetivo específico ou um subconjunto de objetivos, obtendo soluções melhores que as abordagens tradicionais;
- além de implementação relativamente rápida e fácil.

Alguns conceitos elementares em Otimização Multiobjetiva foram utilizados por Rodrigues para avaliar e comparar as soluções geradas pelos Times Assíncronos. Os principais conceitos são os de "dominância" entre soluções e de "solução eficiente", os quais são apresentados a seguir:

Dominância: sejam f_i, $i = 1, 2, \ldots, k$, funções objetivo a serem analisadas, e seja $\mathcal{S}$ o conjunto de todas as soluções factíveis para o problema; dado duas soluções x_a e x_b em $\mathcal{S}$, se $f_i(x_b) \leq f_i(x_a)$ para todo i e se a desigualdade é estrita para algum i, então dizemos que x_b *domina* x_a, o que é denotado por $x_b \succ x_a$.

Soluções que não se dominam duas a duas são chamadas de *soluções equivalentes.*

Veja, na figura 2.10, um gráfico que descreve a relação de dominância entre seis soluções, identificadas por x_i, $i = 1, 2, \ldots, 6$, segundo duas funções objetivos f_1 e f_2. As soluções x_1, x_2 e x_3 são equivalentes e dominam as soluções restantes, x_4, x_5 e x_6. As soluções x_4 e x_5 são equivalentes, sendo que x_4 domina x_6.

Solução Eficiente: uma solução $x_e \in \mathcal{S}$ é dita *eficiente* se e somente se não existe uma outra solução $x \in \mathcal{S}$ tal que $f_i(x) \leq f_i(x_e)$ para todo i, $i = 1, 2, \ldots, k$, e a desigualdade é estrita para algum i.

Em outras palavras, um solução x_e é eficiente se nenhuma outra solução $x \in \mathcal{S}$ domina x_e. Uma solução eficiente também é chamada de *solução não-dominada*, e é conhecida na literatura como solução do **Pareto Ótimo**.

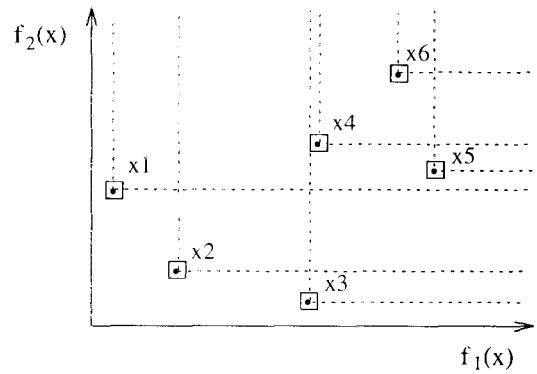


Figura 2.10: Dominância entre soluções.

A meta principal dos algoritmos para Otimização Multiobjetiva, incluindo o uso de Times Assíncronos é, portanto, encontrar soluções do Pareto Ótimo, as quais representam as melhores soluções de compromisso existentes para um determinado problema.

Os Times Assíncronos apresentados por Rodrigues diferem dos times tradicionais na forma como as soluções são organizadas na memória. Cada nova solução produzida pelos agentes é comparada com as outras em termos de dominância e inserida numa estrutura de camadas de soluções equivalentes. Rodrigues introduziu dois novos conceitos para formalizar a organização de soluções em camadas:

Conjunto Não-Dominado: dado um conjunto X de soluções, definimos um *conjunto não-dominado* de X, notação $ND(X)$, como sendo constituído por solucões equivalentes em X que não são dominadas por nenhuma outra solução em X.

Dominância entre Conjuntos Não-Dominados: sejam dois conjuntos não-dominados ND_1 e ND_2; se toda solução de ND_2 for dominada por pelo menos uma solução de ND_1, então dizemos que ND_1 *domina* ND_2 (notação: $ND_1 \succ\succ ND_2$).

Com base nessas definições, a memória é organizada como uma seqüência de camadas de conjuntos não-dominados. Cada camada possui um único conjunto, o qual mantêm uma relação de dominância entre os conjuntos das camadas anteriores e posteriores. Mais especificamente, o melhor conjunto (que domina todos os demais e que não é dominado por nenhum outro) ocupa a primeira posição da memória, camada 1. O conjunto que ocupa a i-ésima posição, camada i, para $1 < i < c$, é dominado pelo conjunto das camadas anteriores $(1, 2, \ldots, i-1)$ e domina os conjuntos das camadas seguintes $(i+1, i+2, \ldots, c)$, com c o número de camadas.

Na figura 2.11, mostramos várias soluções organizadas em conjuntos não-dominados e dispostos em camadas. As soluções pertencentes a um mesmo conjunto estão unidas por uma linha.

A estrutura de camadas é dinâmica, uma vez que as atividades de inserção e remoção de uma solução na memória pode alterar a ordem de dominância entre soluções, fazendo com que novos conjuntos sejam construídos ou que conjuntos antigos sejam desfeitos. Rodrigues apresentou

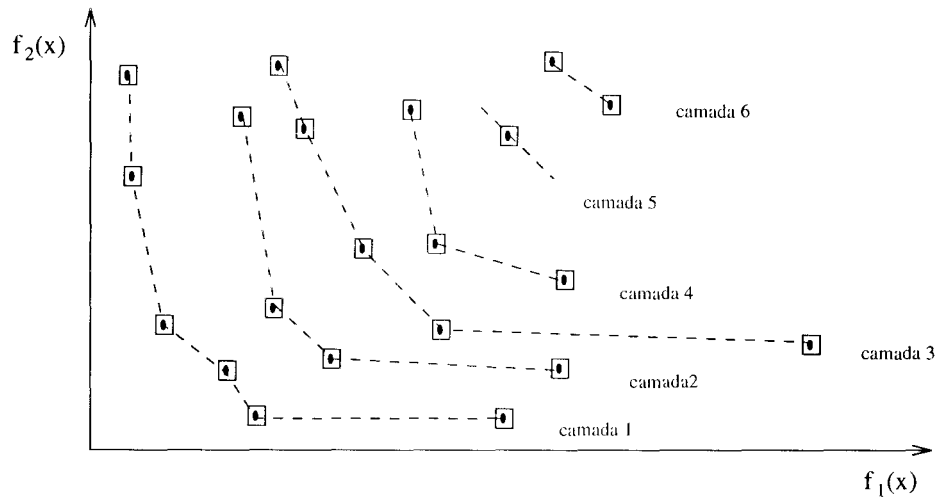


Figura 2.11: Camadas de conjuntos não-dominados.

algoritmos para inserção e remoção de soluções, atualizando as camadas em tempo quadrático no número de soluções na memória. Note que para essas atividades não existe um algoritmo em tempo linear.

A capacidade máxima da memória principal pode ser representada por s soluções distribuídas em c camadas. O maior número de camadas possível é s; caso esse em que cada camada possui uma única solução. Por outro lado, o menor número de camadas é 1, e ocorre quando todas as s soluções são equivalentes, formando um único conjunto não-dominado.

Foi comentado por Rodrigues em [Rod96] que a memória principal do time deve ser grande o suficiente para armazernar o maior número possível das melhores soluções de compromisso (pertencentes ao Pareto Ótimo). Entretanto, dependendo da instância do problema e do número de objetivos a serem satisfeitos, a quantidade de soluções do Pareto Ótimo pode ser elevada, sendo computacionalmente inviável determiná-lo com exatidão (para alguns experimentos, verificou-se que o tamanho do Pareto Ótimo cresce exponencialmente com o tamanho da instância do problema e com o número de objetivos). Neste caso, procura-se obter um conjunto não muito grande de soluções de compromisso que estejam distribuídas uniformemente por todo o espectro do Pareto Ótimo. Um conjunto de soluções com essas características é encontrado fazendo uso de Times Assíncronos.

O acesso às soluções na memória pelos agentes do time ocorre através de políticas de seleção e de destruição, de forma semelhante ao modo descrito nas seções anteriores. A diferença principal é que estas políticas enfocam regras para acesso às camadas. Citamos como exemplo as seguintes políticas: escolher uma camada qualquer, escolher a pior camada (última camada na memória), escolher uma camada com distribuição linear de probabilidades da pior para a melhor, etc. Uma vez escolhida a camada, pode-se tomar qualquer solução da mesma, visto que todas são equivalentes, ou escolher uma solução com base em algum outro critério como, por exemplo, escolher a solução que não seja dominada em pelo menos p objetivos.

Adicionalmente às políticas de destruição de soluções já conhecidas, Rodrigues propôs uma nova política que se baseia em uma distribuição triangular de probabilidades. Essa política obteve resultados melhores para a maioria dos testes realizados, representando um equilíbrio melhor entre o tempo de convergência do time e a diversidade de soluções na memória.

Capítulo 3

Uma Nova Abordagem

Conforme observado no capítulo anterior, existe uma grande variedade de problemas em Desenho de Grafos que são *NP-difíceis* e para os quais podemos encontrar uma gama de algoritmos heurísticos capazes de produzir boas soluções. No entanto, tão importante quanto possuir algoritmos para resolver um determinado problema, é saber como e quando esses algoritmos devem ser utilizados.

Na prática, ao nos depararmos com um problema a ser resolvido e com um conjunto de algoritmos adequados para o mesmo, três alternativas podem ser consideradas:

1. selecionar e aplicar apenas um dos algoritmos;
2. selecionar vários algoritmos e aplicar cada um deles isoladamente; em seguida, identificar a melhor solução obtida;
3. ou selecionar vários algoritmos e combiná-los de tal modo que eles cooperem entre si para produzirem melhores soluções.

Se soubermos antecipadamente qual algoritmo consegue o melhor resultado, então a primeira escolha parece ser a mais vantajosa. Contudo, nem sempre isto é possível. Na maioria das vezes, os resultados dos algoritmos variam de acordo com a instância do problema, não havendo como prevê-los antes de sua execução. Pode ser também que cada algoritmo obtenha uma solução boa, com certas características desejáveis que não são encontradas nas soluções produzidas pelos demais algoritmos.

Já na segunda alternativa, obtemos todas as boas e más soluções que podem ser geradas pelo conjunto de algoritmos. Um inconveniente com esta abordagem é que ela exige um maior esforço computacional, sendo que seleciona apenas as melhores soluções.

A terceira alternativa, por sua vez, é a menos comum na resolução de problemas em Desenho de Grafos. No entanto, ela permite alcançar resultados superiores às outras alternativas; é possível, por exemplo, construir um programa que utilize várias estratégias heurísticas para obter soluções de melhor qualidade, através da união de dois ou mais algoritmos distintos. Esta

idéia nos motivou à utilização de Times Assíncronos para desenhar grafos, o que é uma das principais contribuições deste trabalho.

Muitos problemas em Desenho de Grafos podem ser considerados como Multi-algorítmicos e, por conseqüência, possibilitam uma resolução através de Times Assíncronos. De fato, observamos que:

- em geral, desenhar um grafo sob certos critérios estéticos é *NP-difícil*;
- os conflitos entre critérios estéticos dificultam ainda mais a busca por bons desenhos;
- existe uma grande quantidade de algoritmos heurísticos para desenhar grafos;
- e muitos algoritmos podem ser usados de modo iterativo para melhorar continuamente um desenho.

A combinação de algoritmos para desenhar grafos não é, contudo, uma proposta inédita. Conforme descrevemos na seção 2.2.1, Eades em [Ead92] mostrou que um algoritmo Springs pode ser combinado com um algoritmo radial, visando obter desenhos de árvores livres sem cruzamentos e exibindo simetrias.

No desenho de grafos direcionados, o método de Sugiyama e outros [STT81], constituído de três passos, pode ser visto como uma proposta de se combinar algoritmos diferentes. Em cada passo, uma heurística específica é aplicada para satisfazer um determinado critério estético.

Esses exemplos ilustram as vantagens de se combinar heurísticas em Desenho de Grafos. No entanto, tais propostas possuem limitações, porque empregam seus algoritmos em seqüência e comumente para satisfazer um único critério estético por vez. Em função disto, a execução dos algoritmos é restrita de modo a não "destruir" as características estéticas alcançadas em etapas anteriores. Um outro ponto a ser observado é que essas propostas não foram apresentadas como abordagem geral para combinação de algoritmos, mas sim, como soluções específicas para certos problemas.

O presente trabalho caracteriza-se por utilizar a combinação de algoritmos através de Times Assíncronos como uma nova abordagem para desenhar grafos.

Quando comparada com alguns métodos gerais empregados no desenho de grafos, entre eles, a proposta de Davidson e Harel [DH89] que faz uso de Simulated Annealing, a nossa abordagem destaca-se por analisar os critérios estéticos sem a necessidade de agrupá-los todos em uma única função energética (embora, isto pode ser feito se desejado). O uso de Times Assíncronos também possibilita a obtenção de mais de uma solução de compromisso de modo simultâneo e apresenta um elevado potencial para paralelismo, uma vez que os agentes podem ser distribuídos sobre uma rede de computadores.

Nossa abordagem também difere das que utilizam Algoritmos Genéticos por não haver necessidade de definir operadores de mutação e de permutação que sejam eficientes. As próprias heurísticas para desenho disponíveis na literatura podem ser integradas no time, sem exigir grandes mudanças em seu código. Mais do que isso, os Algoritmos Genéticos consistem em um único algoritmo ao passo que Times Assíncronos são conjuntos de algoritmos que trabalham de modo

autônomo. Sob certos aspectos, um Time Assíncrono pode ser visto como uma "generalização" de um Algoritmo Genético. Uma comparação mais detalhada envolvendo Times Assíncronos, Algoritmos Genéticos e Simulated Annealing é apresentada por Souza em [dS93].

Neste capítulo, descrevemos nossa abordagem, descrevendo a estrutura e o funcionamento de Times Assíncronos voltados ao Desenho de Grafos. Inicialmente, estabelecemos uma representação para as soluções produzidas pelos agentes do time.

Representação das soluções

Uma *solução* para um problema em Desenho de Grafos é constituída das coordenadas de todos os vértices do grafo em estudo, obtidas pela aplicação de uma função do desenho. No caso do padrão gráfico de linhas poligonais, uma solução deve possuir também os dados necessários para representar as dobras das arestas (caso ocorram).

3.1 Estrutura do Time

As memórias, os agentes e o fluxo de dados que compõem a estrutura básica dos Times Assíncronos para desenhar grafos são especificados a seguir:

3.1.1 Memórias

As memórias são formadas por células. Cada célula armazena uma solução (desenho) e informações sobre a qualidade da mesma. O tamanho das memórias é dado pelo número máximo de células que elas possuem. As memórias devem ser grandes o suficiente para permitir uma alta diversidade das soluções.

O processo de iniciação das memórias é a primeira atividade a ser realizada durante a execução do time e as soluções produzidas nesta fase são chamadas de *soluções iniciais*. As memórias podem ser iniciadas de três modos diferentes, com base nas estratégias de preenchimente descritas na seção 2.3.1:

- **Aleatória** - a memória é preenchida com soluções geradas aleatoriamente, escolhendo-se posições quaisquer para os vértices dentro de uma região do plano.
- **Algotimos de Construção** - a memória é preenchida com soluções geradas por algoritmos em Desenho de Grafos.
- **Mista** - parte das soluções é obtida de modo aleatório e parte é gerada por algoritmos de construção.

Essas estratégias apresentam vantangens e desvantagens quanto ao tempo exigido para a convergência e quanto ao grau de diversidade da memória, conforme foi mencionado na seção 2.3.1.

3.1.2 Agentes

Cada agente do Time Assíncrono implementa algum algoritmo para desenho de grafos e utiliza a representação de soluções especificada no início deste capítulo.

A forma mais simples de se projetar um agente consiste em escolher um algoritmo da literatura e anexar ao mesmo um módulo de interface para acesso às memórias, quando então este agente estará pronto para ser adicionado ao time.

É preciso, contudo, tomar algumas precauções, entre elas, evitar que um agente passe muito tempo executando sem acessar as memórias. Caso isto ocorra, o agente não interage suficientemente com os demais e, por conseqüência, contribui pouco para a sinergia no Time Assíncrono.

Quando um algoritmo é muito demorado, torna-se ideal implementar uma versão simplificada do mesmo que realize apenas uma parte do trabalho a que se destina. Um algoritmo iterativo, por exemplo, pode ser codificado de modo a executar algumas poucas iterações por vez. Diz-se, neste caso, que a "granularidade" do algoritmo foi reduzida.

Uma outra forma de modificar um algoritmo para que haja maior sinergia consiste em descarregar na memória as soluções intermediárias produzidas durante a sua execução. O objetivo é fazer com que essas soluções fiquem disponíveis a agentes mais interativos, enquanto agentes mais demorados continuam trabalhando em resultados finais provavelmente melhores.

De um modo geral, deve-se fazer com que o Time Assíncrono produza novas soluções a partir de resultados contidos na memória, o que é conseguido implementando-se agentes de melhoria e de consenso, entre outros. Quando deseja-se incluir no time um algoritmo cujos resultados independem de uma solução inicial, duas escolhas podem ser feitas: utilizar este algoritmo como um agente iniciador ou buscar uma modificação do mesmo que aproveite as características estéticas das soluções já obtidas.

O acesso à memória pelos agentes deve ser delineado por uma política de seleção, de modo semelhante ao discutido na seção 2.3.1. Entre as políticas disponíveis, é possível adotar a escolha de uma solução qualquer dentro de uma "camada" de soluções selecionada ao acaso (a organização das soluções em camadas é explicada mais adiante). Uma outra política pode ser a escolha das melhores soluções na memória. A definição da política de seleção apropriada para cada agente do time é um parâmetro que deve ser configurado de acordo com o problema a ser resolvido.

3.1.3 Fluxo de Dados

A existência de fluxos cíclicos de dados entre as memórias influi nos resultados produzidos pelos agentes, uma vez que garante o "feed-back" do time, tornando possível uma melhoria continua das soluções. Desta forma, faz-se necessário que os agentes sejam organizados de modo a constituírem pelo menos um fluxo cíciclo de dados.

Um modo simples de construir um Time Assíncrono com fluxos cíclicos é definir uma única memória, onde todos os agentes lêem e escrevem soluções. Um diagrama geral de um time para desenho de grafos com esta configuração pode ser visto na figura 3.1.

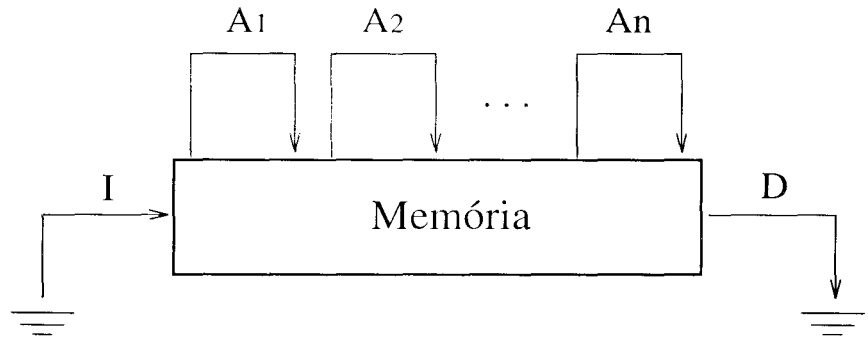


Figura 3.1: Fluxos cíclicos de dados em um Time Assíncrono geral para desenhar grafos.

Osagentes $A_1, A_2, \ldots, A_n$ originam fluxos cíclicos ao acessarem a memória para ler e escrever soluções. O agente I produz as soluções inicias do time, e o agente D é um destruidor de soluções.

3.2 Avaliando a Qualidade das Soluções

Com o objetivo de facilitar a avaliação dos desenhos produzidos pelo Time Assíncrono, deve ser associado a cada solução um vetor que indica a sua qualidade em termos do conjunto de critérios estéticos desejados.

Mais especificamente, sejam $q_{i,G} : \mathcal{S} \rightarrow I\!R$, $i = 1, 2, \ldots, k$, funções de custo que medem a qualidade de uma solução no conjunto $\mathcal{S}$ de todas as soluções possíveis para um grafo G, em relação a k critérios estéticos; altos valores das funções indicam desenhos de pior qualidade no critério estético correspondente; definimos $Q_G : \mathcal{S} \rightarrow I\!R^k$ como uma função que associa a cada elemento x em $\mathcal{S}$, um vetor $Q_G(x) = (q_{1,G}(x), q_{2,G}(x), \ldots, q_{k,G}(x))$ de números reais, que denominamos de *vetor de qualidade*. Quando o grafo estiver subtendido, utilizaremos a notação q_i para indicar o valor da i-ésima coordenada do vetor de qualidade, e Q para indicar o vetor inteiro, omitindo o índice G.

Fazendo uso dos vetores de qualidade e do conceito de dominância explicado na seção 2.3.2 podemos comparar duas soluções e estabelecer qual delas é a "melhor". Deste modo, é possível agrupar soluções em uma estrutura de camadas de conjuntos não-dominados, facilitando a procura pelas melhores ou piores soluções na memória.

A tarefa de computar vetores de qualidade pode ser realizada pelos próprios agentes do time ao produzirem uma nova solução. Cada agente calcula o vetor de qualidade e escreve-o na memória juntamente com a solução correspondente, liberando o agente destruidor desta tarefa. Uma vez na memória, a solução é comparada com as demais pelo agente destruidor, através dos vetores de qualidade, e é incluída em um conjunto não-dominado.

O uso de camadas de conjuntos não-dominados permite obter soluções de compromisso que satisfaçam os critérios estéticos em diferentes proporções, principalmente no caso de existência de conflitos. No entanto, a inclusão e a remoção de uma solução na memória exigem ambas $O(M^2)$ operações (no pior caso) para atualizar os conjuntos não-dominados, onde M é o tamanho da

memória. Visando evitar esta complexidade, que pode ser significativa para memórias contendo muitas soluções e para numerosas iterações dos agentes, elaboramos uma nova forma de organizar as soluções em camadas. A construção das camadas é feita utilizando-se um conceito que introduzimos, chamado de "nível de dominância", e que é definido como segue:

Nível de dominância: seja X um conjunto finito de soluções contido no conjunto $\mathcal{S}$ de todas as soluções factíveis para um determinado problema; definimos o *nível de dominância* de uma solução $x \in X$, dado pela notação μ_x, como sendo o número de soluções em X que dominam x.

As soluções são agrupadas em camadas de acordo com os seus níveis de dominância, de tal modo que soluções de mesmo nível ficam na mesma camada. Além disso, para cada camada i, $1 < i \leq c$, as soluções em i possuem nível de dominância maior que o das soluções na camada $i-1$.

O nível de dominância de uma solução x informa claramente quantas soluções na memória são melhores do que x. Soluções com nível de dominância $\mu = 0$ são as melhores encontradas pelo time, uma vez que nenhuma outra solução obtida possui qualidade superior a elas. Em contra-partida, soluções com alto nível de dominância podem ser consideradas "descartáveis", dado a existência de muitas soluções melhores.

A organização de soluções em conjuntos não-dominados e em níveis de dominância não apresentam uma estrutura idêntica de camadas; soluções que pertecem a uma mesma camada numa abordagem podem ficar em camadas diferentes na outra. Veja um exemplo desta diferença na figura 3.2. No entanto, o melhor conjunto não-dominado é preservado na organização de soluções em níveis de dominância, visto que toda solução do melhor conjunto não-dominado possui nível de dominância igual a 0.

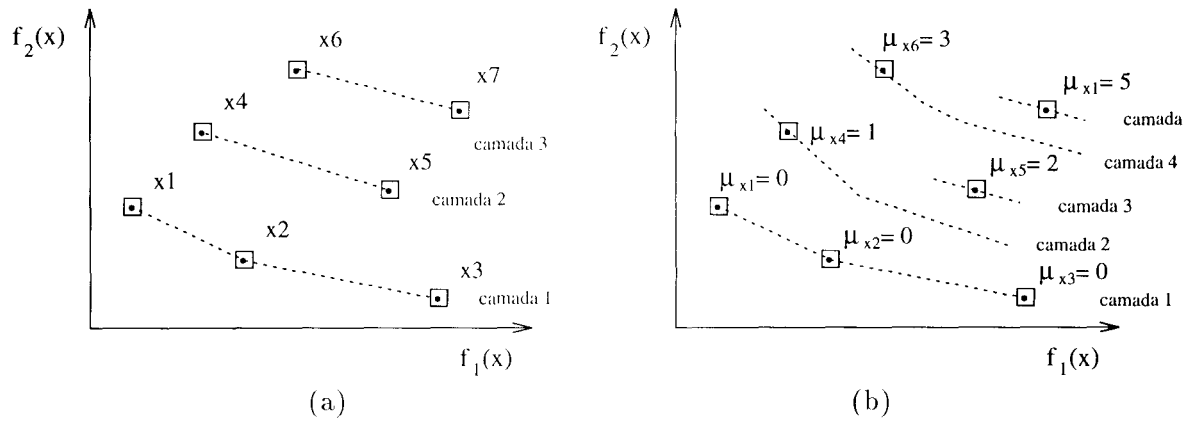


Figura 3.2: Estruturas de camadas de conjuntos não-dominados e de camadas de níveis de dominância.

A principal vantagem do uso de níveis de dominância está no tempo exigido para inserir ou remover uma solução na estrutura de camadas, o qual é linear no número de soluções na memória.

A inserção de uma nova solução envolve basicamente duas atividades: calcular o nível de dominância da nova solução e atualizar o nível de dominância de cada solução antiga da memória. Em seguida, todas as soluções são reunidas em camadas com base nos seus níveis de dominância. Um algoritmo para inserção é apresentado na figura 3.3.

---

1. Seja y a nova solução a ser inserida no conjunto X de soluções na memória, $|X| = M$;
   $\mu_y \leftarrow 0$;
2. **Para** cada $x \in X$ **faça:** { calcule os níveis de dominância das soluções }

   Se x domina y

   $\mu_y \leftarrow Max(\mu_x + 1, \mu_y)$;

   Senão

   Se y domina x

   $\mu_x \leftarrow \mu_x + 1$;
3. $X \leftarrow X \cup \{y\}$;
4. Ordene as soluções em X com base no nível de dominância;
5. Percorra as soluções em X a partir daquela que possui menor nível de dominância até aquela que possui maior nível, agrupando soluções de mesmo nível em uma mesma camada.

Figura 3.3: Algoritmo para inserção de soluções em camadas de níveis de dominância.

Os passos 1 e 3 do algoritmo são realizados em tempo $O(1)$, enquanto que os passos 2 e 5 exigem tempo $O(M)$. O passo 4 pode ser feito em tempo $O(M)$ utilizando o algoritmo **Counting Sort** [CLR95] para ordenar as soluções.

Uma vez que os níveis de dominância são números na faixa de 1 a M, a ordenação é possível em tempo linear através do Counting Sort, ao custo de mais espaço de memória. A complexidade total do algoritmo de inserção é, portanto, $O(M)$.

Ao se remover uma solução, deve-se atualizar os níveis de dominância das soluções que permaneceram na memória e reorganizar estas soluções em camadas. Apresentamos na figura 3.4 um algoritmo para remoção de soluções.

O passo 1 exige tempo $O(1)$. Os passos 3 e 4 são idênticos aos passos 4 e 5 do algoritmo de inserção, respectivamente. Portanto, o algoritmo para remoção também executa em tempo linear, $O(M)$.

O uso de níveis de dominância é uma contribuição importante à estrutura de camadas proposta por Rodrigues em [Rod96], porque demanda menor tempo para inserir e remover soluções na memória, e possibilita analisar os critérios estéticos separadamente, preservando o conjunto das melhores soluções de compromisso alcançado.

1. Seja y a solução a ser removida do conjunto X de soluções na memória, $|X| = M$; $X \leftarrow X - \{y\}$;
2. **Para** cada $x \in X$ **faça**:
   Se y domina x
   $\mu_x \leftarrow \mu_x - 1$;
3. Ordene as soluções em X com base no nível de dominância;
4. Percorra as soluções em X a partir da que possui menor nível até a de maior nível de dominância, agrupando soluções de mesmo nível em uma mesma camada.

Figura 3.4: Algoritmo para remoção de soluções em camadas de níveis de dominância.

Critérios Estéticos Secundários

A análise de critérios estéticos em coordenadas distintas do vetor de qualidade tem por objetivo satisfazer a tais critérios amplamente, sem a necessidade de atribuir pesos para determinar uma importância relativa entre os mesmos, como faz a abordagem de Davidson e Harel em [DH89]. Assim, o time obtém soluções boas para cada critério isoladamente e produz soluções intermediárias de compromisso.

Contudo, nem todos os critérios estéticos devem ser avalidados em coordenadas separadas do vetor de qualidade. Alguns critérios que chamamos de *secundários* somente são desejáveis se forem atendidos em conjunto com outros. Por exemplo, minimizar o número de dobras de arestas em linhas poligonais é interessante apenas quando os desenhos também satisfazem alguns aspectos estéticos como poucos cruzamentos e distribuição uniforme dos vértices. Desse modo, uma função $f : \mathcal{S} \rightarrow I\!R$ que computa o número total de dobras de um desenho não pode ser utilizada sozinha em uma coordenada do vetor, pois permite que certas soluções anômalas sejam consideradas de boa qualidade pelo time. Um exemplo de caso anômalo é uma solução em que todos os vértices estão colapsados em um único ponto do desenho. Tal solução possui um elevado número de cruzamentos e uma má distribuição dos vértices, contudo, não apresenta dobras (uma vez que todas as arestas têm comprimento zero). Esta solução é ótima em número de dobras e seria incluída, por conseqüência, no melhor conjunto de soluções produzidas, até que o time encontrasse uma nova solução sem dobras e com menos cruzamentos ou melhor distribuição de vértices. Mesmo que uma solução melhor exista, em geral, os vértices ficam extremamente próximos um dos outros com o intuito de evitar as dobras. Um modo de resolver este problema consiste em combinar critérios estéticos secundários com critérios não secundários (que chamamos também de *critérios principais*) através de uma ponderação.

Uma forma de se combinar um critério estético secundário, representado por uma função de custo não-negativa z, com r critérios estéticos principais avaliados por funções não-negativas $f_1, f_2, \ldots, f_r$, consiste em definir um vetor de qualidade $(r+1)$-dimensional e as funções:

$$q_i(x) = f_i(x)$$
$$\text{para } i = 1, 2, \ldots, r$$

$$q_{r+1}(x) = z(x) + \lambda \sum_{j=1}^{r} f_j(x)$$

onde q_i informa o i-ésimo elemento do vetor de qualidade, e λ é uma constante escolhida de tal modo que $z(x) << \lambda$ para qualquer solução x.

A função z permite diferenciar soluções com a mesma característica estética nos r critérios estéticos principais e com valores distintos para o critério secundário.

A relação de dominância entre soluções quando tomamos apenas os critérios principais é mantida ao se incluir o critério secundário no vetor de qualidade. Isto significa que o time ainda pode encontrar o mesmo conjunto de soluções de compromisso anterior. A diferença é que essas soluções são agora ligeiramente melhoradas no que diz respeito à satisfação do outro critério estético.

Um ponto a ser observado é que os critérios estéticos secundários não precisam ser combinados com todos os critérios principais, mas apenas com aqueles que são conflitantes aos mesmos.

3.3 Destruição de Soluções

Como já mencionamos antes, a destruição de soluções é uma atividade de fundamental importância pois determina a convergência do time para boas soluções.

O agente destruidor deve implementar políticas de destruição de soluções baseadas na manipulação de camadas. Dependendo do uso do conceito de conjuntos não-dominadas ou de níveis de dominância para organizar as soluções na memória, a operação de destruir uma solução pode ser feita em tempo linear ou quadrático, respectivamente.

Uma nova política de destruição foi desenvolvida para favorecer a busca por muitas soluções de compromisso. A política define a probabilidade $P(i)$ de escolha de uma camada i na memória, $1 \leq i \leq c$, como sendo:

$$P(i) = \frac{i-1}{Tam(i)W} ,$$

onde Tam(i) é uma função que retorna o número de soluções na camada i, e

$$W = \sum_{j=1}^{c} \frac{j-1}{Tam(j)}.$$

Quando todas as camadas têm o mesmo número de soluções, a política se comporta de modo análogo à de distribuição linear de probabilidades. Na medida em que o número de soluções em uma camada aumenta, menor é a chance da mesma ser escolhida. Em um caso extremo, se o número de soluções cresce linearmente da primeira até a última camada na memória, a política estabelece uma distribuição uniforme de probabilidade.

A nova política é adequada ao uso de conjunto não-dominados, pois favorece a permanência na memória de muitas soluções equivalentes para serem melhoradas pelos agentes do time.

3.4 Execução do Time

Antes que a maioria dos agentes comecem a executar, um agente preenche a memoria com soluções iniciais a serem melhoradas. A partir deste momento, todos os demais agentes iniciam o seu trabalho, escolhendo uma solução da memória segundo alguma política de seleção e produzindo novas soluções.

Sempre que o número de soluções na memória atinge um certo limite (por exemplo, o tamanho máximo da memória), um agente destruidor elimina uma solução, abrindo espaço para novas soluções.

Cada agente é responsável por calcular o vetor de qualidade das soluções que produz, reduzindo assim o tempo que o destruidor leva para comparar soluções e decidir quais delas devem ser eliminadas.

O time permanece em execução até que nenhuma mudança ocorra na melhor camada de soluções. Neste momento, a memória do time deve conter bons desenhos do grafo.

3.5 Detalhes de Implementação

Existem muitas formas de se implementar os Times Assíncronos; contudo, enumeramos a seguir três modelos principais que estão relacionados ao modo como agentes podem ser codificados. São eles:

1. **procedimentos de um programa**: agentes são implementados como procedimentos de um programa simples. As memórias do time podem ser implemtadas como uma região de memória acessível a todos os agentes.

2. **threads**: agentes são implementados como linhas de execuções independentes em um programa "multi-thread". Mecanismos de exclusão mútua devem ser empregados para garantir a integridade das informações contidas nas memórias.

3. **programas independentes**: agentes são implementados como programas independentes que acessam uma região de memória compartilhada. Um caso particular de implementação que adota este modelo, utiliza uma abordagem **cliente-servidor** em que os agentes são programas clientes e as memória são disponibilizada por programas servidores.

Entre os três modelos, o primeiro apresenta a vantagem de uma fácil codificação. Entretanto, ele torna extremamente difícil a execução em paralelo de agentes possivelmente concorrentes.

O segundo modelo também é de fácil codificação e permite explorar paralelismo executando o time sobre um sistema operacional multi-thread em uma máquina paralela. Neste modelo, os agentes podem ser "alocados" aos processadores disponíveis na máquina.

No terceiro modelo, a codificação não é tão fácil como nas propostas anteriores, pois exige uma forma mais sofisticada de comunicação entre os agentes e a memória. Além disso, a execução do time demanda maior esforço do Sistema Operacional, com perda significativa de tempo no

escalonamento dos processos. A vantagem deste modelo é que ele permite executar o Time Assíncrono em uma rede de computadores, distribuindo os agentes por máquinas paralelas e/ou convencionais, a fim de atingir um alto grau de paralelismo. Outra vantagem é que novos agentes podem ser adicionados ao time sem a necessidade de mudanças no código dos demais.

Uma outra forma de implementar um Time Assíncrono consiste em utilizar os modelos 2 e 3 em conjunto: os agentes são codificados como threads em um único programa, enquanto as memórias são implementadas como um programa a parte; o acesso às memórias é feito através de uma abordagem cliente-servidor. A grande vantagem deste modelo híbrido é a oportunidade de explorar paralelismo em mais de um máquina paralela, com menos esforço para o Sistema Operacional no escalonamento dos agentes. Isto pode ser feito replicando-se o grupo de agentes sobre um rede de computadores (paralelos ou não).

Capítulo 4

Aplicando a Abordagem

Neste capítulo, exemplificamos a nossa abordagem apresentando um Time Assíncrono para desenhar grafos gerais e um time para grafos direcionados. Os agentes incluídos nos times não são os mais eficientes; no entanto, eles são adequados para mostrar a factibilidade da abordagem proposta.

No final deste capítulo, comentamos ainda sobre algumas configuração da abordagem.

4.1 Um Time Assíncrono para Desenhar Grafos Gerais

Podemos combinar algoritmos Springs, Baricentro e heurísticas aleatórias, de uma maneira simples e flexível em um Time Assíncrono, de modo que cooperem sinergeticamente permitindo desenhar grafos gerais com duas características estéticas básicas, cujos problemas correspondentes são conflitantes e NP-difíceis. O objetivo maior é mostrar como reduzir cruzamentos de arestas e, ao mesmo tempo, conseguir as simetrias proporcionadas pelo método Springs descrito na seção 2.2.4.

Os resultados apresentados aqui são baseados em um trabalho inicial publicado em [dMdNdS96].

4.1.1 Padrão Gráfico e Critérios Estéticos

O padrão gráfico que adotamos para os grafos gerais consiste no desenho de linhas retas, sendo que os vértices são representados como círculos. Este padrão é bastante comum para esa classe de grafos.

Os critérios estéticos que desejamos satisfazer são basicamente dois:

- *máximo número de simetrias* e
- *mínimo número de cruzamentos entre arestas.*

A simetria será aproximada pela função energética do modelo Springs de Kamada e Kawai em [KK88], que define a energia total do sistema de molas como:

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

onde p_i e p_j são as coordenadas dos vértices v_i e v_j do grafo, respectivamente, para $1 \leq i, j \leq n$; l_{ij} é a distância euclidiana desejada entre v_i e v_j, definida por $l_{ij} = L * d_{ij}$, sendo d_{ij} a distância teórica entre os vértices v_i e v_j, e L uma constante que representa o comprimento desejado para as arestas; e o parâmetro k_{ij} é a constante elástica da mola entre v_i e v_j, dado por $k_{ij} = K/d_{ij}^2$, com K uma constante.

A minimização da energia E produz configurações do sistema de molas que estão associadas em geral a desenhos simétricos. Alguns outros critérios, tais como: comprimento uniforme de arestas, distribuição uniforme de vértices e resolução angular, também estão "implícitos" no método Springs.

4.1.2 Descrição do Time

Combinamos o método Springs com heurísticas baseadas no método do Baricentro descrito por Sugiyama em [STT81] visando superar a deficiência do Springs em reduzir cruzamentos. Alguns agentes que provocam uma pertubação aleatória nos desenhos também foram adicionados ao time para aumentar a diversidade de soluções na memória. Note que os agentes que implementamos não eliminam totalmente os cruzamentos, o que é exemplificado mais adiante.

O Time Assíncrono que desenvolvemos possui 8 agentes e uma única memória, como ilustra a figura 4.1. Relembramos o leitor que a memória é representada por um retângulo e os agentes são representados por arcos que entram e/ou saem da memória, denotanto leitura e/ou escrita de uma solução, respectivamente. O nome de identificação de cada agente é colocado junto ao seu arco correspondente.

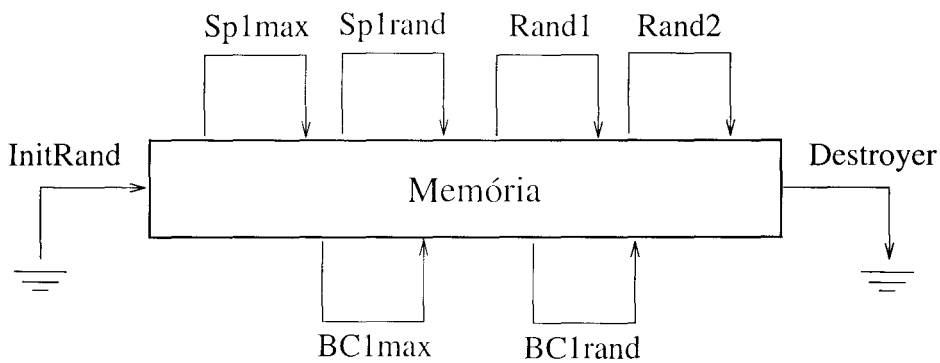


Figura 4.1: Um Time Assíncrono para desenhar grafos gerais.

Os agentes que integram o time são detalhados a seguir:

1. **InitRand** é o agente iniciador do time. Este agente preenche a memória com soluções iniciais, as quais são obtidas escolhendo-se coordenadas para cada vértice de maneira aleatória dentro de uma região retangular.

2. **Sp1max** é uma implementação reduzida do método Springs proposto por Kamada e Kawai. O agente Sp1max lê uma solução da memória e move apenas um dos vértices mais críticos para uma posição que minimiza a energia total do sistema. A nova solução é inserida na memória.

3. **Sp1rand** é semelhante ao agente Sp1max, contudo escolhe um vértice qualquer para ser movido (caso este vértice não esteja em seu mínimo local).

4. **BC1max** é um agente que lê uma solução e descobre o vértice que contribui com o maior número de cruzamento de arestas. Esse vértice é colocado em sua posição baricêntrica com base nas coordenadas x e y de seus adjacentes. A solução resultante é escrita na memória.

5. **BC1rand** é semelhante ao agente BC1max, contudo escolhe ao acaso um vértice a ser movido para a sua posição baricêntrica. A idéia de funcionamento dos agentes BC1max e BC1Rand foi inspirada no método do Baricentro descrito na seção 2.2.3.

6. **Rand1** é um agente que lê uma solução e "pertuba" a localização de um dos vértices escolhido ao acaso. Esse vértice é movido para uma posição aleatória dentro de uma vizinhaça definida por uma região retangular que contenha o desenho. A solução resultante é armazenada na memória.

7. **Rand2** é semelhante ao Rand1, exceto pelo fato de que o vértice é movido dentro de uma vizinhança retangular menor, que contenha o vértice mas não todo o desenho. Os agentes Rand1 e Rand2 não apenas favorecem uma maior diversidade de soluções como também permitem explorar o espaço de soluções na vizinhança da solução atual.

8. **Destroyer** é o agente destruidor de soluções. A *política de destruição* que utilizamos adota uma distribuição linear de probabilidades combinada com o tamanho das camadas de soluções na memória, conforme explicado no capítulo 3. Neste time, o agente destruidor também filtra as novas soluções, de forma a evitar que a memória fique impregnada por desenhos muito parecidos (essa atividade é descrita mais adiante).

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

Em seguida, os demais agentes entram em execução. Estes agentes escolhem as células da memória que serão lidas e trabalham de maneira autônoma e iterativa sobre cópias de seus conteúdos, produzindo novas soluções.

A política de seleção adotada pelos agentes Sp1max e Rand2 consiste em sempre escolher a melhor camada de soluções da memória. Os demais agentes escolhem soluções em camadas quaisquer.

Os resultados produzidos pelos agentes são avaliados associando-se a cada solução um vetor de qualidade bidimensional. O primeiro elemento do vetor informa a energia do sistema de molas, e o segundo indica o número de cruzamentos de arestas. As soluções são organizadas em camadas de conjuntos não-dominados para memórias com até 400 células. Para memórias maiores percebemos que o tempo para inserir ou remover uma solução é elevado (mais do que

o tempo que os agentes levam para produzir a solução) e, portanto, utilizamos o conceito de níveis de dominância. Uma comparação entre os resultados obtidos com o uso de conjuntos não-dominados e com o uso de níveis de dominância é apresentada posteriormente.

No caso específico deste time para grafos gerais, o agente destruidor filtra as novas soluções produzidas, de modo que somente soluções com energia menor ou com cruzamentos "distintos" em relação a alguma outra solução na memória sejam armazanadas. Isso é necessário para evitar o preenchimento da memória com soluções muitas parecidas, uma vez que uma pequena mudança na posição de um vértice pode gerar uma novo desenho com os mesmos cruzamentos e com energia diferente da solução original. O agente destruidor filtra soluções implementando o algoritmo apresentado na figura 4.2:

Seja y a nova solução a ser armazenada na memória;

Se existe uma solução x na memória com os mesmos cruzamentos de y, isto é: as mesmas arestas se cruzam,

Se a energia Springs de y é menor que a energia de x

Elimine x da memória e insira y;

Senão

Descarte y; {não insira y na memória}

Senão

Elimine uma solução da memória utilizando uma política convencional de destruição e insira y;

Figura 4.2: Filtro de soluções.

O teste para verificar se duas soluções possuem os mesmos cruzamentos é feito de forma aproximada comparando-se a soma dos índices das arestas que se cruzam. O número referente a soma desses índices serve para diferenciar as soluções geradas pelos agentes.

Uma vez iniciado, o time permanece em execução até que nenhuma mudança significativa, em termos de simetria e de número de cruzamentos, seja feita no conjunto das melhores soluções contidas na memória (localizadas na camada 1).

4.1.3 Resultados

Nesta seção apresentamos alguns desenhos obtidos pelo time.

A figura 4.3(a) mostra uma solução inicial, produzida pelo agente InitRand. As soluções iniciais são bastante aleatórias, o que proporciona uma grande diversidade de desenhos a serem escolhidos e melhorados.

As figuras 4.3(b), 4.3(c), 4.3(d) e 4.3(e) mostram desenhos do mesmo grafo, produzidos pelo time a partir das soluções iniciais.

A figura 4.3(b) é um desenho típico gerado pelo método Springs. Este desenho possui arestas de comprimento uniforme, exibindo simetrias e expressando a topologia tridimensional do grafo. Já a figura 4.3(c) não possui cruzamentos, e é uma solução impossível de ser obtida através do uso do método Springs isoladamente. De fato, este desenho está associado a um alto valor energético no modelo de molas[1], e a sua obtenção é decorrente da sinergia entre os agentes do Time Assíncrono. A sinergia ocorre porque um agente lê uma solução gerada por algum outro, e a melhora segundo um critério estético que não estava sendo satisfeito. A nova solução pode até ficar pior do que a original em certos critérios, mas é provável que seja melhor no critério para o qual foi trabalhada. Os desenhos 4.3(b) e 4.3(c) representam soluções retiradas da camada 1 (melhor conjunto de soluções obtido), entretanto, a memória contém soluções piores ilustradas pelos desenhos 4.3(d) e 4.3(e). Caso a nova solução e a anterior não se dominem (em seus vetores de qualidade), então elas serão consideradas como soluções de compromisso equivalentes. A seguir fornecemos as respectivas strings que mostram a ordem em que os agentes atuaram sobre as soluções (os números correspondem à numeração dos agentes apresentada na lista com início na página 43).

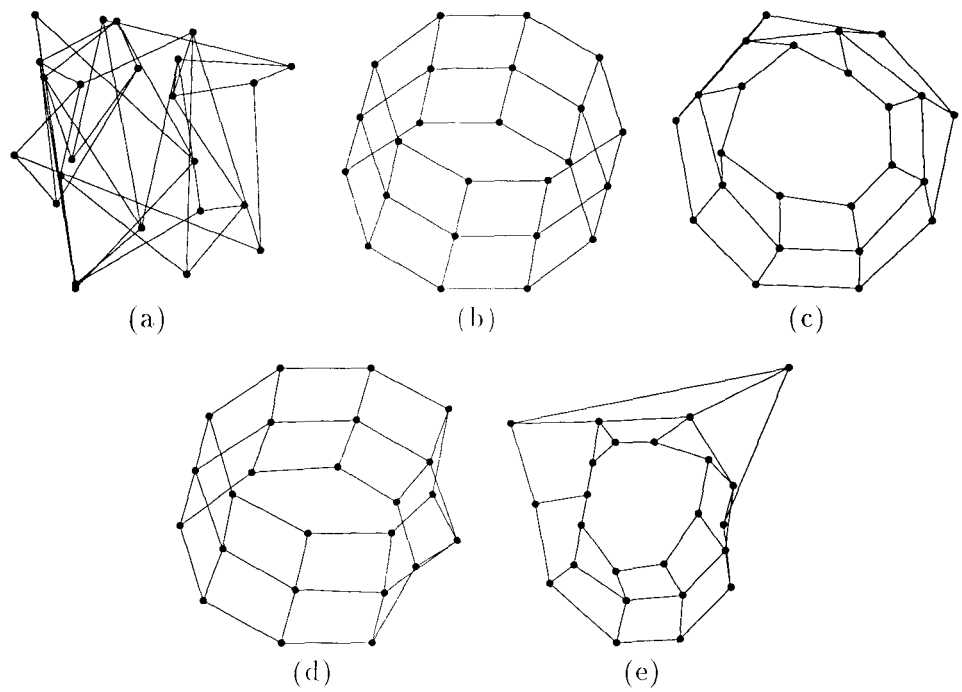


Figura 4.3: Desenhos de uma malha circular.

Solução da figura 4.3(a):
1*

[1]O desenho mostra arestas de comprimentos diferentes, o que significa que algumas molas estão esticadas, enquanto que outras estão comprimidas.

Solução da figura 4.3(b):

144772252227247222222222222772722222222222222222222272227727222222222272222222
222222222222222222222222222222222222222222222222222322222722222222222222222222
222222222222222222222222222222222222222222222222222222222222222222222222222222
222222222222222222222222222222222222222222222222222222222222222222222222222223
222222222222222222222222222222222222222222222222222222222222222222222222222222
222222222222222222232222222222222222222222222222222222222232222222222222222222
222222222222322322322322222222222222232232222222222223*

Solução da figura 4.3(c):

144772252227247222222222222772722222222222222222222272227727222222222272222222
222222222222222222222222222222222222222222222222222322222722222222222222222222
222222222222222222222222222222222222222222222222222222222222222222222222222222
222222222222222222222222222222222222222222222222222722222222222222222222222223
222222222222222222222222222222222222222222223464435643646565643434777477777757747477
777777773777777775737777727577777767747727774772327777772774242277777727 7*

Solução da figura 4.3(d):

144772252227247222222222222772722222222222222222222272227727222222222272222222
222222222222222222222222222222222222222222222222222322222722222222222222222222
222222222222222222222222222222222222222222222222222222222222222222222222222222
222222222222222222222222222222222222222222222222222722222222222222222222222223
222222222222222222222222222222222222223535335333334*

Solução da figura 4.3(e):

144772252227247222222222222772722222222222222222222272227727222222222272222222
222222222222222222222222222222222222222222222222222322222722222222222222222222
222222222222222222222222222222222222222222222222222222222222222222222222222222
222222222222222222222222222222222222222222222222222722222222222222222222222223
222222222222222222222222222222222222222222223464435643646565643434777477777757747477
777777773777777775737777777777777777774773633*

Note que a ordem de execução dos agentes é coincidente com a intuição, por exemplo, na string referente ao desenho 4.3(b) há uma predominância final do agente Sp1max (agente 2); este desenho poderia ser obtido através do método Springs separadamente. Entretanto, no desenho 4.3(c) vemos uma interação maior entre os agentes no final da string, resultando em uma solução que não pode ser obtida pelo método Springs sozinho. As demais strings demosntram que os desenhos 4.3(d) e 4.3(e) são descendentes de alguma das soluções intermediárias (geradas durante o processo, e provavelmente já eliminadas pelo agente Destroyer). Nenhum agente, exceto o agente InitRand, atuou sobre o desenho 4.3(a).

Como outro exemplo, observe, na figura 4.4, seis desenhos do grafo completo K_9, os quais foram obtidos simultaneamente após algum tempo de execução do time. Os desenhos 4.4(a), 4.4(b), 4.4(c), 4.4(d), 4.4(e) e 4.4(f) estão apresentados em ordem decrescente de simetrias e possuem, respectivamente, 106, 94, 68, 52, 48 e 36 cruzamentos. Estes desenhos ilustram como os

critérios estéticos podem ser conflitantes, pois perdemos simetrias na medida em que reduzimos o número de cruzamentos. No entanto, o Time Assíncrono produziu um amplo espectro de soluções de compromisso.

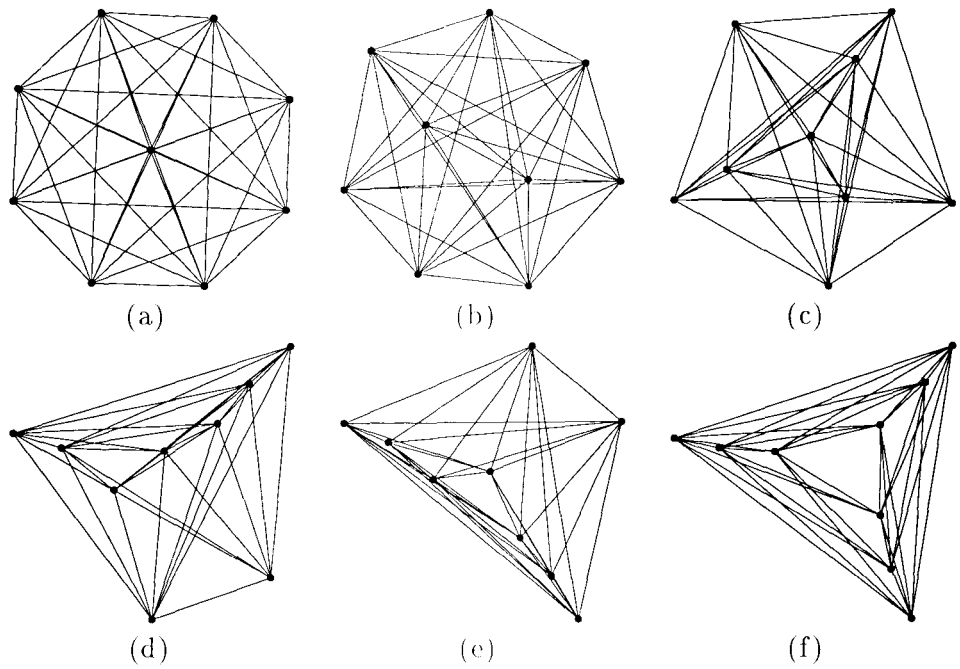


Figura 4.4: Desenhos do grafo completo K_9.

Na figura 4.5 vemos desenhos de um "cubo". O desenho 4.5(b) possui resolução angular ruim, próximas a cruzamentos. Analisando o comportamento do time, verificamos que os agentes tendem a reorganizar os vértices do desenho para que as arestas fiquem com comprimento uniforme, como apresentado na figura 4.5(a). Contudo, qualquer movimento de apenas um vértice por vez resulta em soluções com um número maior de cruzamentos e alta energia inicial, as quais têm grande probabilidade de serem eliminadas da memória. Deste modo, os agentes pioram a resolução angular baixando a energia total da solução, sem criar novos cruzamentos.

Em função dessa característica, certos desenhos com poucos cruzamentos não podem ser melhorados pelo time para exibir simetrias, embora exista a possibilidade de tais soluções, como mostrado na figura 4.5(c).

As figuras 4.6 e 4.7 ilustram outros desenhos obtidos pelo time.

Aperfeiçoamentos do Time

O Time Assíncrono da figura 4.1 nem sempre obtêm uma solução com número mínimo de cruzamentos, mesmo para grafos bastante simples. Por exemplo, as soluções planares para o grafo ilustrado na figura 4.3 são obtidas apenas em algumas execuções[2]. A dificuldade de se

[2]Para os demais grafos apresentados, o time produziu soluções com menor número de cruzamentos em todas as execuções que realizamos.

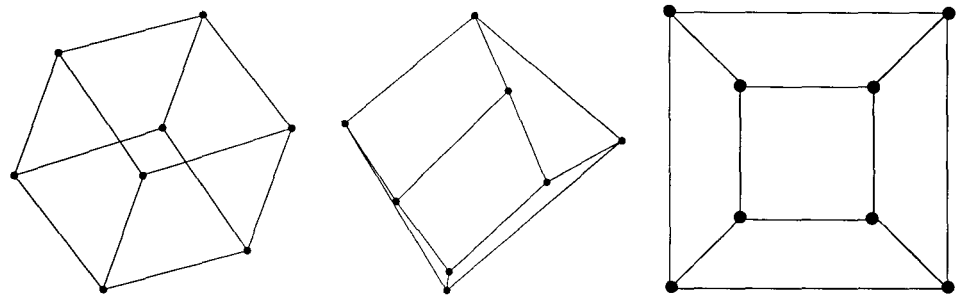


Figura 4.5: Desenhos de um cubo.

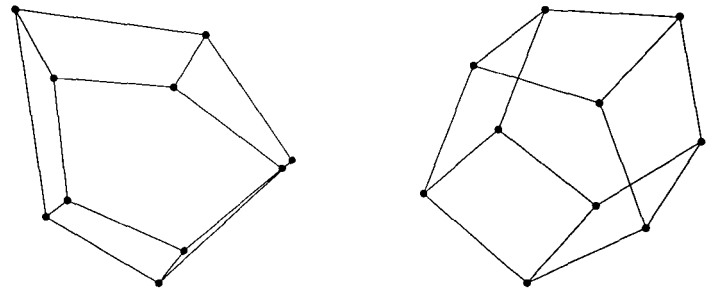


Figura 4.6: Desenhos de uma malha com 5 retângulos.

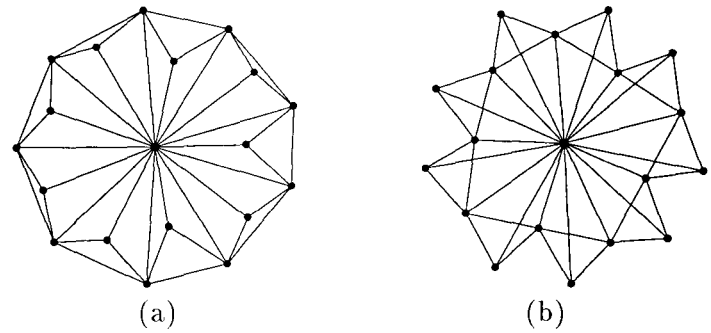


Figura 4.7: Desenhos da Rosa dos Ventos de nove pontas.

obter desenhos melhores deste grafo está associada a uma rápida convergência para a solução 4.3(b) que representa um mínimo local. Os agentes BC1max e BC1rand não conseguem desfazer certos tipos de cruzamentos, como, por exemplo, no caso dessa solução.

A fim de aumentar a capacidade do time em resolver cruzamentos incluímos dois novos agentes, os quais chamamos de "MoveE" e "ReduceE".

O agente MoveE lê uma solução da memória e escolhe aleatoriamente um par de arestas que se cruzam. Uma dessas arestas é movida[3] para uma nova posição, paralela à anterior, de forma a eliminar o cruzamento. Este processo é ilustado na figura 4.8. O agente escreve a nova solução na memória, mesmo que outros cruzamentos tenham sido gerados.

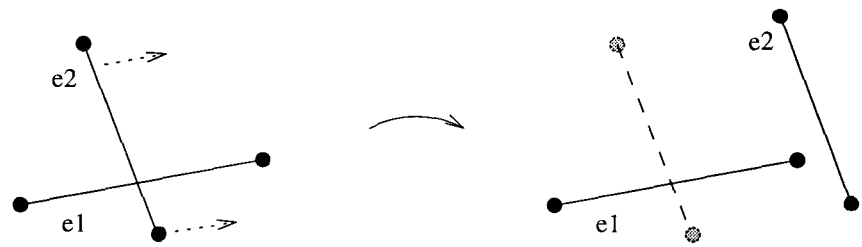


Figura 4.8: Atividade realizada pelo agente MoveE.

O agente ReduceE desfaz cruzamentos reduzindo o comprimento de uma das arestas que se cruzam. O processo consiste em mover um dos vértices extremos de uma aresta para uma posição que elimine o cruzamento, como mostra a figura 4.9.

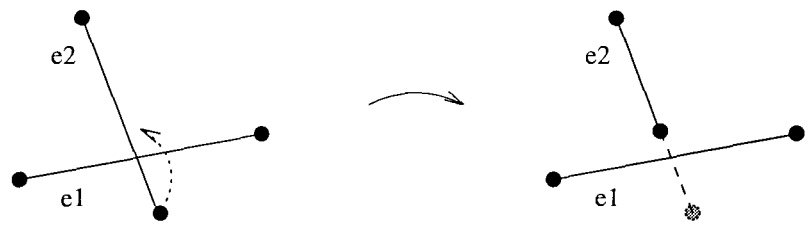


Figura 4.9: Atividade realizada pelo agente ReduceE.

Foram testes comparativos envolvendo o time original da figura 4.1 e o novo time incluindo os novos agentes. Cada time foi executado um certo número de vezes[4] para desenhar o grafo da figura 4.3, e foi verificado quantas execuções obtiveram desenhos planares. O time incluindo os novos agentes obteve desenhos planares do grafo em 70% das execuções, enquanto que o time original obteve desenhos planares em 52% das execuções. A capacidade do time em remover cruzamentos de arestas foi portanto expandida com a utilização dos agentes MoveE e ReduceE.

Uma outra idéia experimentada no time consistiu em utilizar uma nova função energética

[3]Mover uma arestas significa mover os seus vértices extremos.

[4]Realizamos 100 execuções dos times, iniciando a memória com um conjunto fixo de soluções. Testes com mais execuções apresentaram resultados comparativos semelhantes.

que permitisse obter desenhos simétricos com melhor resolução angular, como, por exemplo, Davidson e Harel em [DH89], considerando-se apenas as seguintes parcelas da função de energia:

- critério de distribuição uniforme de vértices,
- critério de comprimento mínimo de arestas e
- critério de distância máxima entre vértices e arestas.

O vetor de qualidade foi redefinido de modo que uma de suas coordenadas representasse uma ponderação desses três critérios. Como resultado, a energia envolvendo a distância entre vértices e arestas melhorou significativamente a resolução angular. Veja na figura 4.10 um desenho do cubo obtido pelo time utilizando a nova energia.

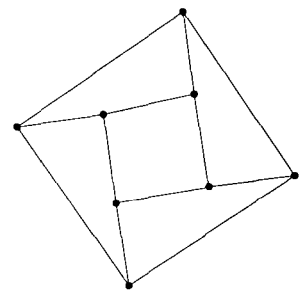


Figura 4.10: Desenho de um cubo sem cruzamentos e com melhor resolução angular.

4.2 Um Time Assíncrono para Desenhar Grafos Direcionados

A mesma abordagem baseada em Times Assíncronos pode ser utilizada para desenhar grafos direcionados, como mostramos nesta seção. Obviamente, outros critérios estéticos e um padrão gráfico diferente devem ser levados em conta.

4.2.1 Padrão Gráfico e Critérios Estéticos

Para o desenho de grafos direcionados simbolizamos os vértices por retângulos contendo um número de identificação, e as arestas por linhas poligonais. Todas as arestas voltadas para baixo são desenhadas como linhas cheias, enquanto que as demais arestas são indicadas por linhas tracejadas. Um caso particular consiste das arestas horizontais, permitidas no time, que são desenhas como linhas cheias e com um seta informado a sua orientação. As arestas que não estão voltadas para baixo são chamadas neste capítulo de *arestas contrárias.*

De modo semelhante ao padrão freqüêntemente adotado na literatura, o desenho é gerado sobre uma grade, o que significa que as coordenadas dos vértices são valores inteiros.

Quando uma aresta conecta vértices em níveis não consecutivos, ela é dividida em arestas menores e vértices falsos são inseridos no encontro da mesma com as linhas que definem os

níveis intermediários, como foi descrito na seção 2.2.3. Este processo leva à construção de um novo grafo $G' = (V', E')$, a partir do grafo original $G = (V, E)$, onde V' é o conjunto V acrescido dos vértices falsos, e E' é o conjunto E em que arestas longas foram substituídas por arestas menores resultantes da inseção de vértices falsos.

Os critérios estéticos que desejamos satisfazer são os seguintes:

1. *minimizar o número de arestas contrárias* (obter orientação uniforme),
2. *minimizar o número de cruzamentos entre arestas*,
3. *minimizar o comprimento das arestas*,
4. *minimizar o número de dobras das arestas* e
5. *minimizar o número de vértices falsos.*

Os critérios estéticos principais para avaliar a qualidade dos desenhos são os de número 1 e 2. Os critérios 3, 4 e 5 são secundários e devem ser combinados em uma ponderação com os critérios principais, no intuito de impedir a produção de soluções anômalas.

Impomos uma restrição ao time que consiste em evitar que dois vértices ocupem uma mesma posição sobre um nível. Quando isto ocorre, alguns vértices são movidos para uma nova posição à direita ou à esquerda, abrindo espaço para distribuir os vértices sobrepostos.

4.2.2 Descrição do Time

O time é formado por uma memória e doze agentes como mostra a figura 4.11.

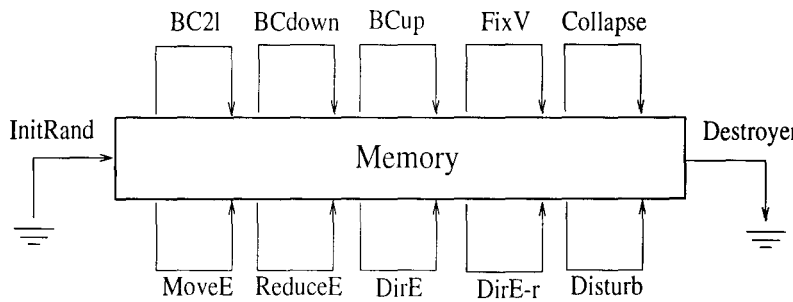


Figura 4.11: Um Time Assíncrono para desenhar grafos direcionados.

A função de cada agente é apresentada abaixo.

1. **InitRand** é o agente iniciador. Este agente preenche a memória com soluções iniciais que são geradas de forma aleatória.
2. **BC2l** implementa uma versão simplificada do método do Baricentro descrito na seção 2.2.3 para dois níveis. O agente BC2l lê uma solução da memória e escolhe aleatoriamente dois níveis consecutivos que representamos aqui por l_i e l_{i+1}. A ordem dos vértices do

nível l_i é fixada, e o Baricentro é aplicado sobre o nível l_{i+1}. Em seguida, este processo é repetido fixando-se l_{i+1} e aplicando o Baricentro para o vértices do nível l_i. A nova solução produzida é escrita na memória.

3. **BCdown** aplica o Baricentro sobre cada dois níveis consecutivos, partindo do nível mais alto até o mais baixo, mantendo sempre o nível superior fixo e aplicando o Baricentro para o nível inferior. Quando o agente atinge o nível mais baixo, a nova solução é escrita na memória.

4. **BCup** é idêntico ao BCdown, exceto pelo fato de trabalhar partindo do nível mais baixo até o mais alto, fixando o nível inferior e aplicando o baricentro no nível superior.

5. **FixV** escolhe um vértice ao acaso e tenta movê-lo no mesmo nível para uma posição livre que reduza cruzamentos ou dobras das arestas. Se tal posição não existir, um outro vértice é escolhido. O agente conclui uma solução quando um vértice é movido ou quando todos os vértices foram testados. Se um vértice foi movido, o agente escreve a solução modificada na memória.

6. **Collapse** "colapsa" dois níveis consecutivos, reduzindo assim o comprimento das arestas originais. Esta operação ocasiona uma contração das arestas que possuem os seus extremos em cada um dos níveis a serem colapsados e pelo menos um destes extremos é um vértice falso. No caso de arestas em que ambos os extremos são vértices reais, elas serão colocadas na horizontal (com os seus vértices num mesmo nível). Se houver necessidade, este agente move vértices à direita ou à esquerda, de forma que 2 vértices não ocupem a mesma posição. A solução produzida é armazenada na memória.

7. **MoveE** escolhe aleatoriamente um par de arestas que se cruzam (se existir algum) e move uma dessas arestas para uma nova posição que elimine o cruzamento, de modo semelhante ao agente MoveE desenvolvido para grafos gerais. A aresta a ser movida é deslocada sobre o eixo formado pelos vértices extremos da outra aresta, sem modificar, contudo, a sua inclinação. O agente escreve a solução na memória, mesmo que novos cruzamentos tenham sido gerados.

8. **ReduceE** escolhe aleatoriamente um par de arestas que se cruzam (se existir) e muda um de seus extremos para uma posição elimina o cruzamento. A solução gerada é escrita na memória.

9. **DirE** lê uma solução e verifica se ela possui arestas contrárias. Caso afirmativo, escolhe uma dessas arestas e inverte a sua orientação através da seguinte regra: sejam l_u e l_v, respectivamente, os níveis dos vértices extremos u e v de uma aresta contrária; o vértice u é movido para o nível $l_v + 1$ ou o vértice v é movido para o nível $l_u - 1$, com escolha feita ao acaso. O agente escreve a solução resultante na memória.

10. **DirE-r** é uma versão mais complexa do agente DirE. Ele escolhe uma aresta contrária (u, v) (se existir alguma), com l_u e l_v os níveis dos vértices u e v, respectivamente, e move

todos os vértices do conjunto alcançável R_v de v para $\mid l_v - l_u \mid +1$ níveis abaixo de sua posição. O vértice u não é movido, mesmo que ele pertença a R_v. O agente escreve a solução na memória.

11. **Disturb** escolhe um vértice qualquer e o move aleatoriamente para uma posição dentro da região retangular do desenho. A solução é escrita na memória.

12. **Destroyer** elimina soluções da memória segundo a nova política de destruição discutida na seção 3.3.

Todos os agentes adotam uma política de seleção que determina a escolha de soluções em uma camada qualquer da memória.

As soluções produzidas pelos agentes são comparadas umas com as outras através de vetores de qualidade de 4 coordenadas. As funções q_j, para $j = 1, 2, 5, 4$, associadas às coordenadas de um vetor qualidade, são definidas como mostramos abaixo:

$$q_1 = f_1,$$

$$q_2 = f_2,$$

$$q_3 = \lambda * (f_1 + f_2 + 1) + f_3,$$

$$q_4 = \lambda * (f_1 + f_2 + 1) + f_4 + f_5,$$

onde f_i, $i = 1, 2, \ldots, 5$, são funções que determinam a qualidade de uma solução em cada um dos 5 critérios estéticos especificados na seção anterior, respectivamente; e λ é uma constante estritamente maior que $f_3(x)$ e $f_4(x) + f_5(x)$ para toda solução x no conjunto $\mathcal{S}$ de soluções factíveis para o problema.

Os critérios estéticos 4 e 5 foram analisados em uma mesma coordenada do vetor de qualidade, pois são depedentes entre si. Na verdade, as curvas nas arestas (representadas pelo critério 4) ocorrem quando alguns vértices falsos não produzem uma linha reta com os seus adjacentes nos níveis imediatamente acima e abaixo.

Na próxima seção, apresentamos alguns desenhos gerados pelo Time Assíncrono. A maioria dos grafos que utilizamos foram retirados de uma lista de artigos sobre desenho de grafos direcionados.

4.2.3 Resultados

A figura 4.12 mostra alguns desenhos para pequenas modificações de um grafo encontrado em [SM91]. A figura 4.12(a) é o desenho original do grafo; as demais figuras são desenhos obtidos pelo time para um grafo modificado. A mudança feita no grafo consistiu na inserção do vértice de número 8, que não aparece na figura original.

Os desenhos iniciais produzidos pelo time possuem muitos cruzamentos e arestas contrárias. Na medida em que os agentes vão trabalhando, um conjunto de soluções melhores é obtido. Ao final do processo, o time encontra um amplo conjunto de soluções, que varia entre desenhos

com poucos cruzamentos e desenhos com poucas arestas contrárias, como ilustramos nas figuras 4.12(b), 4.12(c) and 4.12(d).

A figura 4.12(b) tem 13 cruzamentos e 1 aresta voltada para cima (7 arestas contrárias, se considerarmos as que possuem orientação horinzontal). A figura 4.12(d) tem 4 cruzamentos e 4 arestas voltadas para cima (4 arestas contrárias ao todo). O time encontra muitas soluções intermediárias como mostrado na figura 4.12(c), com 8 cruzamentos e 3 arestas voltadas para cima (3 arestas contrárias).

O critério estético de orientação uniforme é analisado pelo Time Assíncrono estabelecendo-se penalidades diferentes para as arestas contrárias. Mais especificamente, a penalidade de uma aresta horizontal é menor que a de uma aresta voltada para cima.

A Figura 4.13 mostra desenhos obtidos pelo time para o Forrester's World Dynamics Graph. Esse é um grafo popularmente usado em artigos sobre desenho de estruturas hierárquicos, como em [TDB88, GM89, For71]. A Figura 4.13(a) tem 54 cruzamentos e 6 arestas voltadas para cima (8 arestas contrárias), a figura 4.13(b) tem 47 cruzamentos e 6 arestas voltadas para cima (11 arestas contrárias), a figura 4.13(c) tem 34 cruzamentos e 7 arestas voltadas para cima 10 arestas contrárias), e a figura 4.13(d) tem 27 cruzamentos e 8 arestas voltadas para cima (13 arestas contrárias).

O último exemplo é um desenho de um grafo que expressa a árvore geneológica da família de sistemas Unix. Este grafo pode ser desenhado pelo Time Assíncrono sem cruzamentos e sem arestas contrárias como mostramos na figura 4.14. A abordagem original de Sugyiama não consegue um desenho planar deste grafo [STT81].

Aperfeiçoamentos do Time

O time pode ser aperfeiçoado adicionando-se novos agentes para obter soluções melhores nos critérios estéticos. Um possível agente consiste em escolher duas arestas que se cruzam e trocar a ordem de seus vértices com relação aos níveis onde estão posicionados. Este processo é feito a partir do ponto em que ocorre o cruzamento, avançando para os níveis acima ou abaixo até que se alcance um vértice extremo à uma das arestas. Tal agente foi implementado e consegue resolver alguns cruzamento simples, como aqueles exibidos pelas arestas localizadas mais à direita nas figuras 4.13(a), 4.13(b) e 4.13(c).

Outra melhoria no time também é conseguida modificando-se as funções que avaliam os critérios estéticos. Experimentamos, por exemplo, definir pesos para as dobras das arestas de modo a penalizar dobras que ocorrem em pontos distantes de vertices extremos. Com isso, arestas muito longas tendem a ficar mais retas no meio, facilitando a visualização do grafo.

4.3 Características da Abordagem

Comentamos, a seguir, alguns aspectos de configuração dos Times Assíncronos para desenho de grafos.

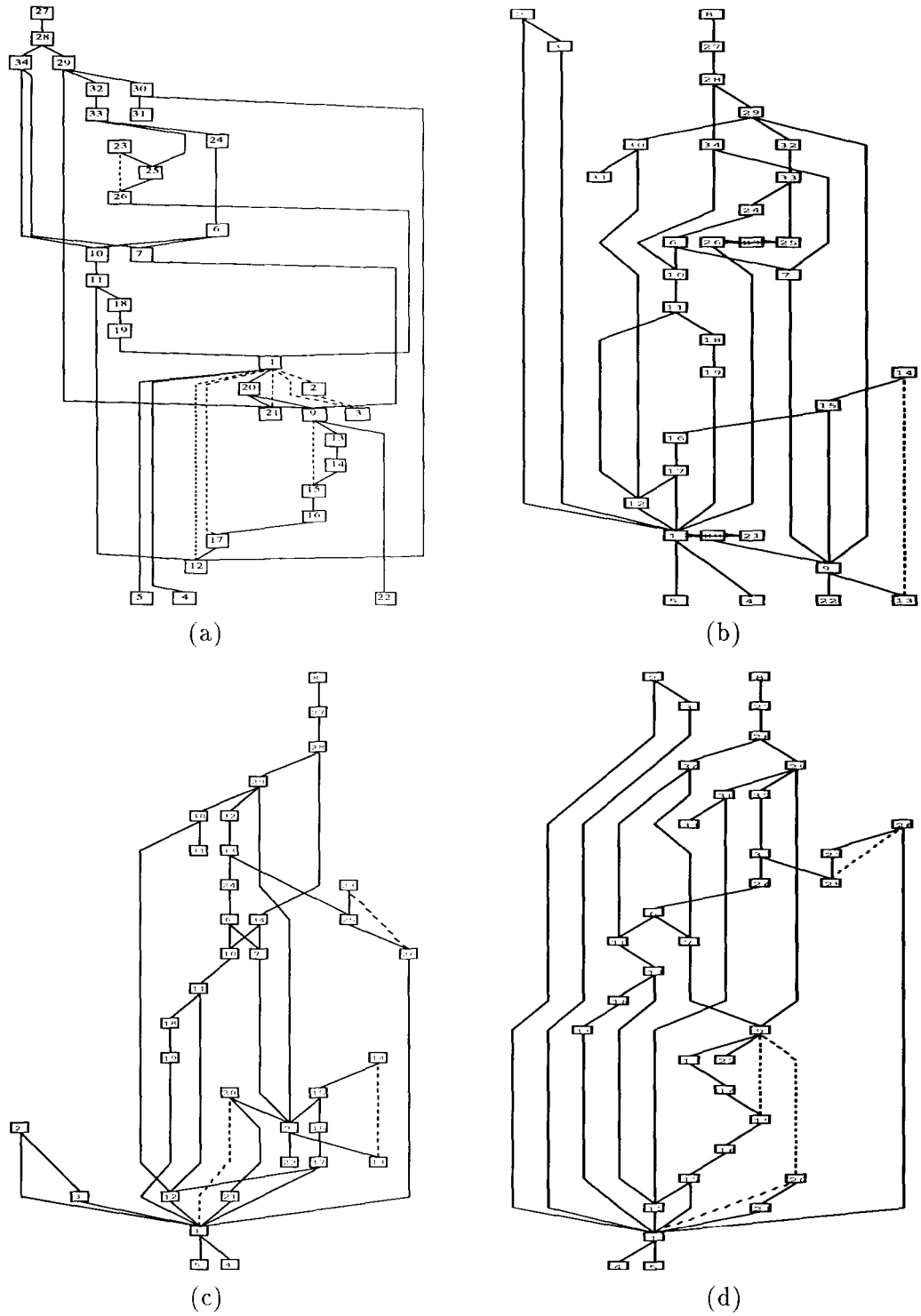


Figura 4.12: Desenhos de um grafo que expressa a sintaxe da linguagem C.

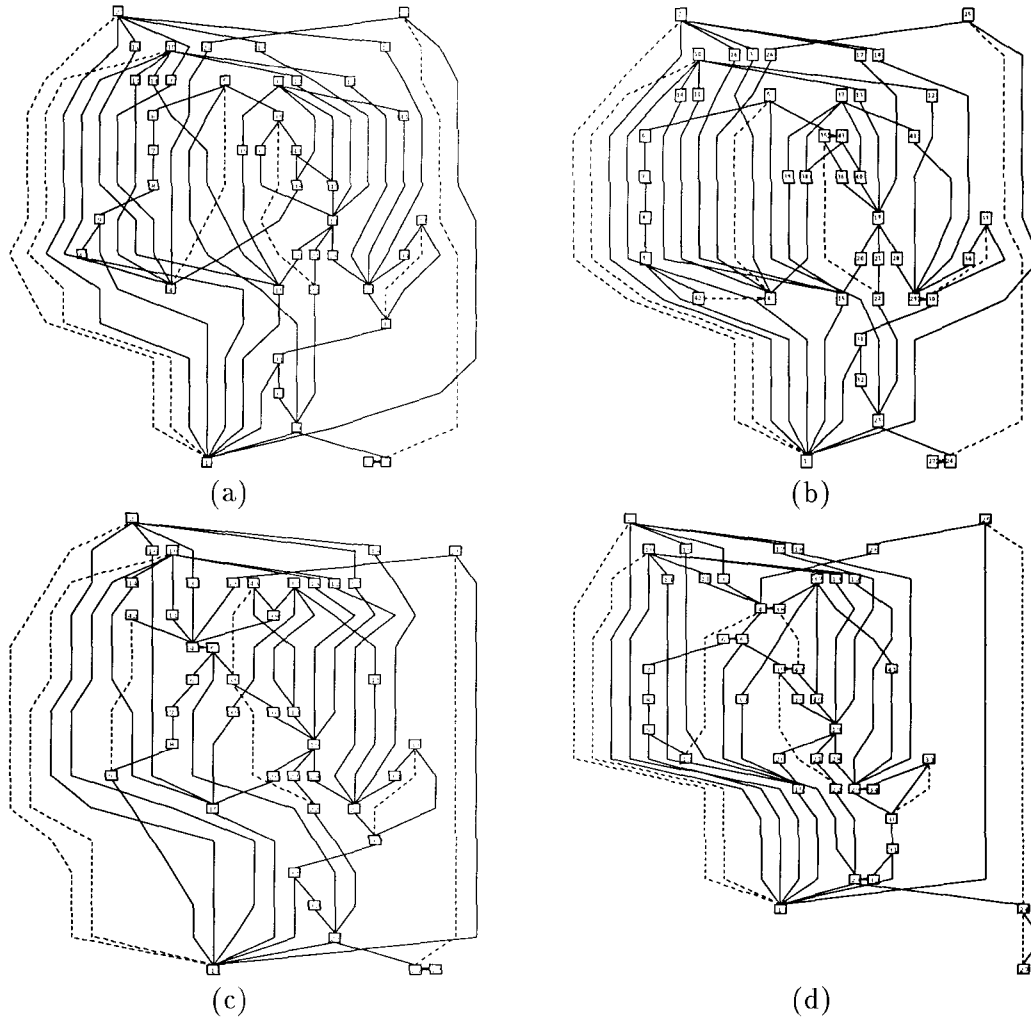


Figura 4.13: Desenhos do Forrester's World Dynamics Graph.

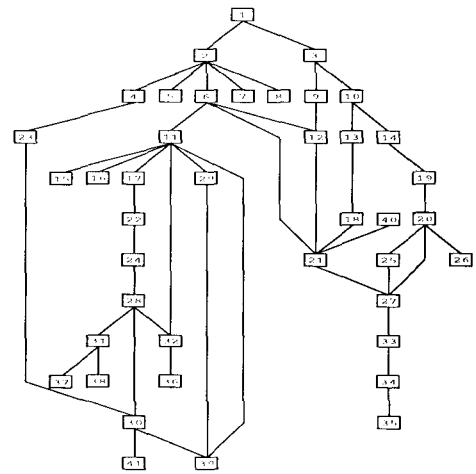


Figura 4.14: Desenho da árvore genealógica da família de sistemas Unix

Diferenciação de Soluções

Um aspecto importante no desenvolvimento de um Time Assíncrono para desenhar grafos é estabelecer o que diferencia uma solução das outras. Isto é necessário pois desejamos evitar que soluções idênticas sejam armazenadas na memória.

A forma que escolhemos para diferenciar soluções foi compará-las pelos vetores de qualidade. Consideramos que duas soluções são diferentes se os seus vetores de qualidade são distintos em pelo menos uma coordenada.

Em alguns casos comparamos também as soluções através da verificação das arestas que se cruzam; soluções que apresentam cruzamentos distintos são tomadas como diferentes, mesmo que possuam vetores de qualidade idênticos. Um caso em que essa diferenciação é útil ocorre quando o vetor de qualidade representa um único critério estético geral como, por exemplo, orientação uniforme ou mínimo número de cruzamentos. Estes critérios permitem muitas soluções boas de mesmo vetor de qualidade, que devem ser diferenciadas a fim de possibilitar a sua inclusão na memória. A verificação de cruzamentos é uma alternativa viável para diferenciar soluções.

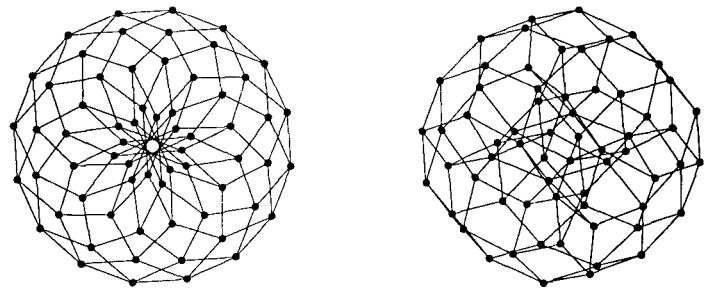
Sobre o Tamanho da Memória

Comentamos no capítulo anterior que o tamanho das memórias deve ser proporcional ao tamanho do grafo a ser desenhado. Em particular, estabelecemos o tamanho das memórias como $k * |G|$, onde k é a dimensão dos vetores de qualidade associados às soluções, e $|G|$ é tamanho do grafo. Essa relação foi escolhida empiricamente e mostrou-se suficiente para maioria dos grafos que desenhamos. Valores maiores para as memórias, em geral, aumentam o tempo de execução sem apresentar melhoria significativa da sinergia do time.

Sobre os Agentes Incorporados aos Times Assíncronos

Visando aumentar a interação entre os agentes, e conseqüentemente a sinergia do time, muitos algoritmos foram implementados em uma forma reduzida, executando um número limitado de iterações por vez. Verificamos, contudo, que o time de algoritmos reduzidos nem sempre é capaz de produzir os mesmos desenhos que aqueles times que possuem algoritmos em sua versão completa. Por exemplo, o método Springs completo obtêm, em algumas execuções, o desenho de um toro mostrado na figura 4.15(a), ao passo que o Time Assíncrono formado pelos agentes Spl max e Spl rand parece sempre cair no mínimo local de maior energia ilustrado na figura 4.15(b). A explicação para este fato é que as soluções produzidas por um agente são modificadas pelos demais, o que interfere no seu processo individual de busca por desenhos de qualidade. Em particular, o método Springs é bastante instável, pois pequenas mudanças na posição de alguns vértices direciona o sistema para mínimos locais distintos. Deste modo, pode ser interessante incluir no time algoritmos completos ou executá-los em separado a fim de que obtenham desenhos sem a interferência dos outros algoritmos.

Verificamos também que um Time Assíncrono formado por agentes simples demora para produzir bons desenhos nos critérios estéticos escolhidos. Em geral o conjunto de soluções evolui



(a) (b)

Figura 4.15: Desenhos de um toro.

muito lentamente por dois motivos: são necessárias várias iterações dos agentes e um vetor de qualidade deve ser computado para cada nova solução obtida. Além disso, quando os agentes não implementam nenhuma estratégia de longo prazo para melhorar as soluções na memória, o time pode gerar resultados diferentes em outras execuções. Estas características foram observadas particularmente para o desenho de grafos direcionados com muitos vértices e arestas.

Sobre os Critérios Estéticos

Quando modelamos um critério estético através de uma função energética, uma pequena mudança na posição de um vértice pode originar uma solução diferente. Caso isso ocorra, a nova solução é armazenada na memória, mesmo que seja numericamente parecida com as demais.

Um problema que ocorre quando a memória fica cheia de soluções extremamente parecidas é a redução da diversidade. Este problema foi percebido durante o desenvolvimento do time para grafos gerais, sendo resolvido atráves da utilização de um filtro, como descrevemos na seção 4.1.2.

Deve ser observado, contudo, que o filtro pode dificultar a saída do time de um mínimo local, uma vez que descarta soluções com energia pior em relação a alguma outra solução na memória.

O uso do filtro funcionou bem no caso da energia do sistema Springs, entretanto, não deve ser considerado como uma altenativa geral para qualquer função energética.

Camadas de Soluções e Políticas de Destruição

Comparamos o uso de conjuntos não-dominados com a proposta baseada em níveis de dominância para organizar as soluções da memória em camadas. Como era previsto, a abordagem de níveis de dominância exigiu tempo menor para inserir ou remover uma solução.

Quanto à capacidade de obtenção de bons desenhos, o desempenho das abordagens variou de acordo com a política de destruição utilizada. Testamos 3 políticas de destruição, cujos resultados podem ser vistos na tabela 4.1. Esta tabela mostra a capacidade do time de grafos gerais em obter desenhos planares do grafo de malha circular, ilustrado na figura 4.3.

Observe que a proposta de níveis de dominância mostrou-se um pouco melhor que a aquela que utiliza conjuntos não-dominados, para as políticas de destruição baseadas em distribuição

Política de destruição	% das execuções que obtiveram desenhos planares	
	Conjuntos não-dominados	Níveis de dominância
distribuição linear	26%	42%
distribuição triangular	4%	6%
nova política (distribuição linear combinada com o tamanho das camadas)	52%	32%

Tabela 4.1: Teste das políticas de destruição envolvendo as abordagens para organizar soluções em camadas (uso de camadas de conjuntos não-dominados e de camadas de níveis de dominância.

linear e em distribuição triangular. No entanto, o time que implementa conjuntos não-dominados e que adota a nova política de destruição obteve desenhos planares em uma porcentagem maior de execuções.

Note também que a política de destruição baseada em distribuição triangular de probabilidade não se mostrou muito eficaz. Verificamos que ela favoreceu a produção de soluções de má qualidade nas últimas camadas da memória, consumindo espaço que poderia ser utilizado para armazenar soluções melhores.

Tempo de Resposta

Para obter as figuras dos grafos gerais apresentadas neste capítulo, cada instância do time levou entre 5 a 15 minutos, executando em um computador PC PENTIUM 200Mhz, com 32Mb de memória RAM, sob sistema operacional Linux. Para o caso de grafos direcionados, necessitamos cerca de 120 minutos de processamento, sendo que, em algumas instâncias, esse tempo baixou para 40 minutos[5].

Em muitas aplicações que desenham grafos exigindo interação com o usuário (como nas interfaces visuais), o tempo limite disponível para produzir um desenho é questão de segundos. Frente a essa característica, a abordagem baseada em Times Assíncronos apresenta um tempo de computação elevado e pode ser inadequada para algumas atividades. No entanto, o nosso objetivo é desenhar grafos encontrando um conjunto de boas soluções de compromisso que satisfaçam critérios estéticos *NP-difíceis* e conflitantes, o que justifica uma maior tolerância quanto ao tempo de processamento. Além disso, é possível implementar otimizações que agilizem a execução dos Times Assíncronos. Um tipo de otimização consiste em calcular o vetor de qualidade de uma nova solução aproveitando as informações disponíveis na solução original, lida da memória. A idéia é recomputar apenas a qualidade estética de parte do desenho que foi modificado. Outro modo de reduzir o tempo de processamento seria explorando paralelismo através de um dos modelos de implementação descritos no capítulo 3.

Vale dizer que implementamos os Times Assíncronos com a finalidade maior de demonstrar a possibilidade de desenhar grafos através da combinação de heurísticas. Não houve, por

[5] A diferença de tempo de execução entre os times para grafos gerais e grafos direcionados deve-se à própria natureza do problema: no desenho de grafos direcionados, manipulamos adicionalmente vértices falsos e também atendemos a um número maior de critérios estéticos.

conseqüência, preocupação em desenvolver otimizações no código, o que significa que os agentes recalculam o número total de cruzamentos e verificam integralmente os demais critérios estéticas para cada nova solução gerada.

Quanto ao modelo de implementação utilizado para codificar os times, optamos pelo enfoque de Programas Independentes. A escolha deste modelo foi determinada pela facilidade em configurar e experimentar diversas estruturas de times, sem a necessidade de efetuar mudanças no código.

Embora o modelo de Programas Independentes permita explorar paralelismo e distribuição, os times foram executados em uma única máquina convencional, com os processos "agentes" e "memória" concorrentes.

Capítulo 5

Conclusão

Avaliação da Abordagem

A abordagem baseada em Times Assíncronos atinge sinergia, fazendo com que os agentes cooperem para obter desenhos melhores que se fossem executados sozinhos. Dependendo do grafo, os times produzem não apenas uma única solução, mas muitas soluções de compromisso que satisfazem os critérios estéticos em proporções diferentes.

A abordagem mostrou-se flexível, uma vez que os vetores de qualidade podem ser modificados com facilidade para refletir novos critérios estéticos, e a idéia pode ser aplicada para desenhar diversas classes de grafos.

Quando confrontado com abordagens tradicionais para Desenho de Grafos, o uso de Times Assíncronos destaca-se pelas seguintes razões: possibilita combinar as características estéticas inerentes a várias estratégias de desenho; pode ser estendido pela inclusão de novos agentes; é flexível; produz um espectro de boas soluções; e permite explorar paralelismo e distribuição dos agentes.

Em contra-partida, a abordagem exige maior recurso computacional envolvendo gastos com memória e tempo de processamento. A exigência de memória é devida ao fato dos times manipularem simultaneamente muitas soluções. O tempo elevado deve-se à execução contínua dos diversos agentes e ao "overhead" causado pela necessidade de se computar os vetores de qualidade.

Uma das dificuldades encontradas na aplicação da abordagem foi o desenvolvimento de agentes eficientes de consenso, isto é, agentes que combinem duas ou mais soluções da memória. Não foi possivel projetar agentes de consenso que fossem capazes de reconhecer parte de duas soluções que são de boa qualidade e, depois, combiná-las em uma nova solução. Sentimos muita dificuldade em determinar o que é bom dentro de uma solução, e também, em tratar problemas de escalamento e de rotação das soluções. A título de experimento, foi projetado um agente que combina ao acaso parte de duas soluções, no entanto, a inclusão deste agente no time não aumentou significativamente a sua sinergia.

Contribuições da Dissertação

A principal contribuição deste trabalho consiste na abordagem que descreve como heurísticas em Desenho de Grafos podem ser combinadas de modo a obterem sinergia.

Outra contribuição importante é a proposta de agrupar soluções utilizando níveis de dominância, a qual possibilita obter um espectro de soluções equivalentes, com tempo linear no tamanho da memória para inserir ou remover uma solução na estrutura de camadas.

Finalmente, o trabalho apresenta um nova política de destruição de soluções que combina distribuição linear de probabilidade com tamanho das camadas, favorecendo a produção de um conjunto maior de soluções de compromisso.

Sugestões para Trabalhos Futuros

Existe um vasto número de aspectos ainda não estudados quanto a combinação de heurísticas de desenho por meio de Times Assíncronos. Em especial, outras configurações de times podem ser experimentadas, utilizando-se novos algoritmos ou um fluxo cíclico de dados diferente. Por exemplo, pode-se fazer uso da técnica de dividir e conquistar para construir um desenho de um grafo partindo de desenhos de seus subgrafos. Já existem alguns estudos nesse sentido [GM89], os quais podem servir como base para o desenvolvimento de novos agentes e para a especificação de uma estrutura de memórias de soluções parciais.

Uma outra proposta consiste em escolher uma nova representação para as soluções que não esteja associada às coordenadas dos vértices no plano. O objetivo é abstrair as características estéticas de maior interesse, permitindo produzir desenhos sem se deter em problemas de escalamento ou de rotação das figuras.

Além disso, uma investigação sobre a nova política de destruição apresentada e sobre o conceito de níveis de dominância deve ser feita a fim de comprovar sua eficiência em obter bons desenhos para outros grafos e critérios estéticos. A nova política e o conceito de nível de dominância também podem ser testados em outras aplicações de Times Assíncronos, não obrigatoriamente em Desenho de Grafos.

Finalmente, seria de grande interesse explorar diversas formas de paralelismo dos times, pois implementamos apenas o modelo de programas independentes baseado em uma abordagem cliente-servidor. O modelo multi-threads é um forte candidato a investigação, uma vez que máquinas paralelas são cada vez mais comuns. Naturalmente, o modelo híbrido oferece vantagens ainda maiores.

Bibliografia

[AK89] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons Ltd., 1989.

[BBS96] J. Branke, F. Bucher, and H. Schmeck. Using genetic algorithms for drawing undirected graphs. Technical Report 347, Institut fu Angewandte Informatik und Formale Beschreibungsverfahren, D-76128 Karlsruhe, Germany, 1996.

[BC87] S. Bhatt and S. Cosmadakis. The complexity of minimizing wire lengths in VLSI layouts. *Inform. Process. Lett.*, 25:263–267, 1987.

[BL76] K. Booth and G. Lueker. Testing for the consecutive ones property interval graphs and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13:335–379, 1976.

[Bol78] Béla Bollobás. *Extremal Graph Theory.* Academic Press, New York, 1978.

[Bra88] F. J. Brandenburg. Nice drawings of graphs and trees are computationally hard. Technical Report MIP-8820, Fakultat fur Mathematik und Informatik, Univ. Passau, 1988.

[Cav95] V. F. Cavalcante. Times assíncronos para o job shop scheduling problem: Heurísticas de construções. M.Sc. thesis, Instituto de Matemática, Estatística e Ciência da Computação, Universidade Estadual de Campinas, Campinas - SP, Brazil, 1995.

[Che93] C. L. Chen. *Bayesian Nets and A-Teams for Power System Fault Diagnosis.* Ph.D. dissertation, Department of Electrical and Computer Engineering of Carnegie Mellon University, April 1993.

[CLR95] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, Mass., 1995.

[DETT94] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.

[DH89] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. Technical report, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 1989.

[Dji95] H. N. Djidjev. A linear algorithm for the maximal planar subgraph problem. In *Proc. 4th Workshop Algorithms Data Struct.*, Lecture Notes in Computer Science. Springer-Verlag, 1995.

[dMdNdS96] C. F. X. de Mendonça, H. A. D. do Nascimento, and P. S. de Souza. Sinergia em desenho de grafos usando springs e pequenas Heurísticas. In *Proc. XXIII Seminario Integrado de Software e Hardware (SEMISH 96)*, Agosto 1996.

[dMN94] C. F. Xavier de Mendonça Neto. *A Layout System for Information System Diagrams.* Ph.D. dissertation, Department of Computer Science, University of Queensland, Australia, March 1994.

[dS93] P. S. de Souza. *Asynchronous Organizations for Multi-Algorithm Problems.* Ph.D. dissertation, Department of Electrical and Computer Engineering of Carnegie Mellon University, April 1993.

[dST91] 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.

[dST93] 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.

[DT89] G. Di Battista and R. Tamassia. Incremental planarity testing. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 436–441, 1989.

[Ead84] P. Eades. A heuristic for graph drawing. *Congr. Numer.*, 42:149–160, 1984.

[Ead89] P. Eades. Complexity issues in drawing directed graphs. In *Proceedings of the International Workshop on Discrete Algorithms and Complexity*, pages 9–15, Fukuoka, Japan, November 1989. Institute of Electronics, Information and Communication Engineers (IEICE), Tokyo.

[Ead92] P. D. Eades. Drawing free trees. *Bulletin of the Institute for Combinatorics and its Applications*, 5:10–36, 1992.

[ET76] S. Even and R. E. Tarjan. Computing an st-numbering. *Theoret. Comput. Sci.*, 2:339–344, 1976.

[EW] P. Eades and N. Wormald. Edge crossings in drawings of bipartite graphs. Technical Report 108, Department of Computer Science, University of Queensland. to appear in Algorithmica.

[EW90] P. Eades and N. Wormald. Fixed edge length graph drawing is NP-hard. *Discrete Appl. Math.*, 28:111–134, 1990.

[Far48] I. Fary. On straight lines representation of planar graphs. *Acta Sci. Math. Szeged.*, 11:229–233, 1948.

[For71] J. W. Forrester. *World Dynamics.* Wright-Allen, Cambridge, Mass., 1971.

[Gar95] A. Garg. On drawing angle graphs. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 84–95. Springer-Verlag, 1995.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.

[GJ83] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM J. Algebraic Discrete Methods*, 4(3):312–316, 1983.

[GM89] D. J. Gschwind and T. P. Murtagh. A recursive algorithm for drawing hierarchical directed graphs. Technical Report CS-89-02, Department of Computer Science, Williams College, 1989.

[Gol89] D. E. Goldberg. *Genetic Algorithm in Search, Optimization, and Machine Learning.* Addison Wesley Publishing Company, 1989.

[Gre89] A. Gregori. Unit length embedding of binary trees on a square grid. *Inform. Process. Lett.*, 31:167–172, 1989.

[GT95a] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 286–297. Springer-Verlag, 1995.

[GT95b] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. Submitted to *SIAM Journal on Computing*, 1995.

[HPYM80] C. L. Hwang, S. R. Paidy, K. Yoon, and A. S. M. Masud. *Mathematical Programming with Multiple Objectives: A Tutorial*, volume 7, pages 5–31. Great Britain: Pergamon Press, 1980.

[HT74] J. Hopcroft and R. E. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[HT93] J. Cai X. Han and R. E. Tarjan. An $O(m \log n)$-time algorithm for the maximal subgraph problem. *SIAM J. Comput.*, 22:1142–1162, 1993.

[KA96] J. Knopman and J. S. Aude. Paralelização de algoritmos genéticos aplicados ao problema de placement em clusters de estações de trabalho. In *Proc. of VIII Simpósio Brasileiro de Arquitetura de Computadores e Processamento de Alto Desempenho*, Agosto 1996.

[KJV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, v. 220(4598):pp. 671–680, May 1983.

[KK88] T. Kamada and S. Kawai. Automatic display of network structures for human understanding. Technical Report 88-007, Department of Information Science, University of Tokyo, 1988.

[KK89] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inform. Process. Lett.*, 31:7–15, 1989.

[La 94] J. A. La Poutré. Alpha-algorithms for incremental planarity testing. In *Proc. 26th Annu. ACM Sympos. Theory Comput.*, pages 706–715, 1994.

[LEC67] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs: Internat. Symposium (Rome 1966)*, pages 215–232, New York, 1967. Gordon and Breach.

[Lin92] X. Lin. *Analysis of Algorithms for Drawing Graphs.* PhD thesis, Department of Computer Science, University of Queensland, 1992.

[Man90] J. Manning. *Geometric Symmetry in Graphs.* Ph.D. Thesis, Department of Computer Sciences, Purdue University, December 1990.

[Man91] J. Manning. Computational complexity of geometric symmetry detection in graphs. Lecture Notes in Computer Science 507:1-7. Springer-Verlag, June 1991.

[MRR+53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal Chem. Physics*, (21), 1953.

[Mur92] S. Murthy. *Synergy in Cooperation Agents: Designing Manipulators from Task Specifications.* Ph.D. dissertation, Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA, 1992.

[Pei95] 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.

[RdS95] R. F. Rodrigues and 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.

[Rod96] 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, 1996.

[RT81] E. Reingold and J. Tilford. Tidier drawing of trees. *IEEE Trans. Softw. Eng.*, SE-7(2):223–228, 1981.

[SM91] K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Trans. Softw. Eng.*, 21(4):876–892, 1991.

[SR83] K. J. Supowit and E. M. Reingold. The complexity of drawing trees nicely. *Acta Inform.*, 18:377–392, 1983.

[ST81] M. N. S. Swamy and K. Thulasiraman. *Graphs, Networks, and Algorithms.* John Wiley & Sons, New York, 1981.

[Ste51] S. K. Stein. Convex maps. *Proc. Amer. Math. Soc.*, 2:464–466, 1951.

[STT81] 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.

[TDB88] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, SMC-18(1):61–79, 1988.

[TdS90] 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.

[TdS92] S. N. Talukdar and P. S. de Souza. Scale efficienty organizations. *IEEE Int. Conference on Systems, Man and Cybernetics*, October 1992.

[Tun93] D. Tunkelang. A layout algorithm for undirected graphs. In *Proc. of the ALCOM International Workshop on Graph Drawing*, pages 14–16, 1993.

[Wag36] K. Wagner. Bemerkungen zum vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.

[WS79] C. Wetherell and A. Shannon. Tidy drawing of trees. *IEEE Trans. Softw. Eng.*, SE-5(5):514–520, 1979.