UNICAMP

Departamento de Engenharia de Computação e Automação Industrial

Faculdade de Engenharia Elétrica e de Computação

Universidade Estadual de Campinas

Relatório de Projeto

de Iniciação Científica

Geração automática de diagramas unifilares

(processo 2005/04748-3)

ALUNO: Paulo César M. Meira

Orientadora: Wu, Shin - Ting

Período do Relatório: 01/03/2006 – 01/08/2006

Agosto de 2006

1 Resumo do Plano Inicial

Diagramas unifilares constituem uma das mais importantes ferramentas de apoio aos engenheiros de potência tanto para planejamento como para monitoramento de uma rede elétrica. A rápida evolução dos recursos gráficos e das técnicas de visualização assistidas por computador despertou o interesse pelos algoritmos de geração automática de diagramas inteligíveis desde o final da década de 90.

O principal objetivo do trabalho é implementar, a partir dos dados de conectividade e dos dados geográficos de uma rede, um algoritmo de geração automática de diagramas unifilares proposto e avaliar o desempenho deste algoritmo em termos de

1. tratamento de conexões cíclicas;
2. tratamento de informações geográficas incompletas;
3. tratamento de redes de médio e grande porte;
4. legibilidade dos diagramas em diferentes níveis de resolução.

Um outro objetivo secundário é integrar o algoritmo no ambiente do VisciPower. Para isso, é necessário familiarizar-se com o VisciPower. Propomos para isso ainda uma terceira tarefa que seria reimplementar as funções de janelas sobre uma outra biblioteca de recursos gráficos mais flexíveis que possibilitariam implementar interações mais complexas com os diagramas unifilares.

Para atingirmos o nosso objetivo propomos o seguinte plano de trabalho:

T1: familiarização com o ambiente VisciPower: migração das funcionalidades da biblioteca GLUI para wxWidgets.

T2: estudo dos algoritmos de geração de diagramas unifilares.

T3: implementação dos algoritmos.

T4: levantamento junto com os engenheiros de potência dos critérios de legibilidade para um diagrama unifilar.

T5: avaliação comparativa dos algoritmos.

T6: integração de um dos algoritmos em VisciPower.

T7: testes de usabilidade da nova interface de VisciPower.

T8: documentação.

Foi ainda elaborado um cronograma preliminar para a realização destas tarefas no período de março de 2006 a fevereiro de 2007.

	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez	Jan	Fev
T1	X	X										
T2	X	X	X									
T3			X	X	X	X						
T4			X	X	X							
T5					X	X	X					
T6								X	X	X		
T7					X	X				X	X	X
T8		X	X	X	X	X	X	X	X	X	X	X

2 Resumo das Atividades Desenvolvidas

De acordo com o cronograma proposto, foram realizadas as seguintes atividades no período a que se refere este relatório:

- Migração do VisciPower de GLUI para wxWidgets
- Estudo e Implementação dos Algoritmos
- Levantamento dos Critérios de Legibilidade

Segue-se uma breve descrição de cada uma destas tarefas, cujo detalhamento, assim como imagens e análises para alguns casos testes em variadas situações. pode ser encontrado no *site* do projeto [1].

2.1 Migração do VisciPower de GLUI para wxWidgets

Adaptação das funções baseadas em OpenGL [8]: as principais funções de desenho existentes no VisciPower [12] foram adaptadas para que pudessem ser utilizadas de modo independente do resto da interface gráfica. Algumas também tiveram adição de parâmetros para torná-las reutilizáveis.

Estudo da biblioteca wxWidgets: o estudo da wxWidgets [15] ocorreu inicialmente através dos tutoriais disponíveis na própria documentação da biblioteca. Quando foi alçancado um nível suficiente de conhecimento, passou-se a consultar a documentação de cada classe específica. Uma classe que mereceu atenção extra foi a classe de interação com OpenGL.

Estudo da biblioteca FTGL [5]: o VisciPower utilizava as funções de desenho de texto por *bitmap* da biblioteca GLUT [4]. Como não há equivalente em wxWidgets, ou seja, não há funções de alto nível, foi necessária a utilização de uma biblioteca para a renderização de texto. De um pequeno grupo disponível gratuitamente e de código livre, a mais atualizada é a biblioteca FTGL. O estudo foi simples e consistiu principalmente na análise dos exemplos que fazem parte da distribuição da biblioteca, que são bem documentados.

Criação da organização gráfica das janelas: inicialmente foi utilizado um programa específico para projeto visual das janelas. No entanto, devido à falta de maturidade do projeto (wxGlade [14]), optou-se por abandoná-lo após o desenvolvimento das primeiras versões do programa de testes e passar a editar o código fonte diretamente, sem auxílio de outros programas. O modelo de organização das janelas do novo VisciPower tenta seguir com fidelidade o modelo adotado na versão original, com alguns aprimoramentos no tratamento de eventos.

Gráficos das funções de demanda e de carregamento: a implementação original dos gráficos de evolução da demanda e do carregamento no tempo era muito restrita e não permitia redimensionamento dos gráficos. Os elementos dos gráficos eram posicionados absolutamente, o que dificultou a criação da nova versão. No entanto, após algum esforço, chegou-se a uma versão de gráficos que podem ser redimensionados de acordo com a necessidade do usuário e que apresenta algumas melhorias visuais em relação à versão original.

Tratamento dos eventos associados a cada componente visual: para a associação dos eventos foi necessário seguir um processo cansativo de busca através do código fonte do VisciPower, pois a biblioteca GLUI [6] não define padrões de codificação, além do código original do programa usar C. Os eventos foram criados para cada componente seguindo o padrão de macros da wxWidgets e tentou-se sempre que possível usar código em C++ e evitar treços em C puramente.

Migração de MySQL [7] para PostgreSQL [9]: migração dos dados de teste armazenados no banco de dados MySQL para o banco PostgreSQL por este último dispor de uma extensão para processamento de dados geo-referenciados.

Migração de Meschach [20] para uBLAS [11]: para testar as funcionalidades de interatividade, foi desenvolvido um simples algoritmo de fluxo de carga no VisciPower. Na sua versão original, este algoritmo foi implementado com uso das funções da biblioteca numérica Meschach, que foi substituída pelo pacote uBlas do projeto Boost [2]. A Meschach é escrita em C mas não é mais atualizada há muito tempo, enquanto a uBLAS é um projeto em constante atualização e é escrita em C++. Além disso, Boost contém outros módulos de funções que utilizaremos ao longo do nosso projeto, o que pode reduzir o número de suas dependências aos pacotes externos. Portar o código de Meschach para uBLAS foi simples, já que são simples funções numéricas.

2.2 Estudo e implementação dos algoritmos

Estudo inicial dos 3 algoritmos: Consistiu numa leitura de todos os artigos encontrados sobre os três algoritmos que propusemos implementar: o algoritmo de Fruchterman [18], o algoritmo de Mota e outros [17] e o algoritmo de Rao e Deekshit [19]. Os dois primeiros algoritmos são baseados em analogia a partículas sujeitas a forças enquanto o terceiro utiliza um posicionamento hierárquico dos nós da árvore de distribuição que o sistema representa. Cada algoritmo foi estudado um por vez para que fosse possível estabelecer uma organização geral do futuro código em C++. Este estudo permitiu avaliar melhor quais bibliotecas seriam utilizadas para implementação do programa de testes.

Verificação da disponibilidade de bibliotecas: Existem algumas bibliotecas de código aberto para trabalho com grafos, sendo a mais geral e promissora a Boost Graph Library (BGL [16]), que também tem uma licença bastante permissiva. Outras bibliotecas existem, mas algumas não apresentam os recursos necessários ou mesmo têm uma licença muito restritiva.

Estudo da Boost Graph Library: o estudo da Boost Graph Library ocorreu através da leitura do livro que descreve seu funcionamento e também contém uma descrição geral das bibliotecas Boost e de programação genérica, na qual a maioria do código desta biblioteca é baseado. Paralelamente foi feita uma revisão de conceitos de programação com *templates* de C++.

Implementações:

Implementação dos algoritmos baseados em forças: inicialmente foram implementados os dois algoritmos baseados em forças, pois a implementação das estruturas de armazenamento dos dados e mesmo cada um dos passos dos algoritmos são bastante parecidos.

Implementação de recursos extras para o ambiente de testes: implementadas as versões iniciais de parte dos algoritmos, percebeu-se que seria necessário implementar alguns recursos extras no programa de testes, incluindo edição, navegação e a possibilidade de carregar e salvar arquivos em alguns formatos. Utilizou-se funções de wxWidgets, OpenGL e também mais um item da Boost, a Boost Tokenizer [3] (visto que os arquivos lidos são texto plano com a formatação específica).

Implementação do algoritmo para circuitos de distribuição: este algoritmo requer estruturas extras e sua implementação é bem diferente dos outros dois. A implementação inicial seguiu as recomendações dos autores do algoritmo, mas em seguida foram efetuadas alterações para adequação do funcionamento. O algoritmo original não prevê o tratamento de conexões cíclicas, por exemplo, e a implementação atual permite ao algoritmo trabalhar com circuitos com tais conexões.

Testes: Os resultados dos testes dos algoritmos implementados foram sintetizados num artigo submetido ao Workshop de Trabalhos de Iniciação Científica em Computação Gráfica, Processamento de Imagens e Visão Computacional (WICCGPI [13]), cuja cópia se encontra no Apêndice A.

Testes simples e ajustes: foram realizados vários testes simples para testar o funcionamento do programa de testes em geral, principalmente sua estabilidade. Através da geração de grafos aleatórios, foi possível corrigir alguns descuidos presentes no código, assim como ajustar melhor os parâmetros das funções de bibliotecas de terceiros.

Testes (diagramas sem nós fixos): nesta etapa deu-se atenção especial à aplicação dos algoritmos a circuitos sem nenhum nó com posição fixa. Foram realizados inúmeros testes, inicialmente com grafos aleatórios e depois com circuitos reais ou fictícios disponíveis como casos teste para estudo de fluxo de potência [10]. Foi possível observar as principais características de cada algoritmo.

Testes (diagramas com nós fixos): o estudo do comportamento dos algoritmos quando alimentados com dados que contém posições fixas foi realizado fixando-se um pequeno número de nós em relação ao total de nós do circuito. Alguns resultados seguem o padrão dos resultados para circuitos sem nós fixos, mas outros representam características dos algoritmos ainda não observadas. O algoritmo para redes de distribuição não foi aplicado à esses conjuntos, pois não prevê o tratamento de nós fixos.

2.3 Levantamento dos Critérios de Legibilidade

Conversando com professores da área de sistemas elétricos e revisando os trabalhos sobre os algoritmos, constatou-se que não existe um estudo formal sobre os critérios de legibilidade específicos

de diagramas unifilares. No entanto, alguns critérios parecem ser aceitos em consenso:

- **Número de cruzamentos de linhas:** é provavelmente o critério mais aceito de todos. Cruzamentos requerem atenção redobrada ao interpretar um diagrama, podendo torná-lo confuso ou mesmo ilegível caso ocorrem em grande número.

- **Distribuição dos nós:** em geral é desejável que os nós estejam distribuídos uniformemente na área de desenho. Observou-se que para um conjunto com algumas centenas de nós a uniformidade total pode ser prejudicial, visto que não é possível analisar todo o diagrama sem utilizar recursos de *zoom* e navegação.

- **Comprimento das linhas:** também é desejável uniformidade no comprimento visual das linhas, mas variações que não sejam extremas são geralmente toleráveis.

Estes critérios aparentam servir de uma base para a análise dos resultados obtidos através dos algoritmos, mesmo constatado que não são absolutos entre os engenheiros e nem podem ser interpretados da mesma forma para qualquer sistema.

3 Plano de Trabalho e Cronograma para o Próximo Período

No período em análise deste relatório, as tarefas propostas no cronograma original foram realizadas, a divisão original do tempo para cada tarefa tem sido satisfatória e cada tarefa vem sendo desenvolvida de acordo com o esperado. Desta forma, não vemos necessidade em alterar o cronograma ou o plano de trabalho propostos anteriormente.

Referências

[1] Geração automática de diagramas unifilares. http://www.dca.fee.unicamp.br/projects/vdx/meira/.

[2] Boost C++ Libraries. http://www.boost.org/.

[3] Boost Tokenizer. http://www.boost.org/libs/tokenizer/index.html.

[4] FreeGLUT. http://freeglut.sourceforge.net/.

[5] FTGL - OpenGL font library. http://homepages.paradise.net.nz/henryj/code/.

[6] GLUI User Interface Library. http://glui.sourceforge.net/.

[7] MySQL. http://www.mysql.com/.

[8] OpenGL - The Industry Standard for High Performance Graphics. http://www.opengl.org/.

[9] PostgreSQL. http://www.postgresql.org/.

[10] Power Systems Test Case Archive. http://www.ee.washington.edu/research/pstca/.

[11] uBLAS: Boost Basic Linear Algebra. http://www.boost.org/libs/numeric/ublas/doc/index.htm.

[12] VisciPower. http://www.dca.fee.unicamp.br/projects/vdx/siqueira/.

[13] WICCGPI - Sibgrapi 2006. http://www.sibgrapi.ufam.edu.br/.

[14] wxGlade: a GUI builder for wxWidgets/wxPython. http://wxglade.sourceforge.net/.

[15] wxWidgets Cross-Platform GUI Library. http://www.wxwidgets.org/.

[16] *The boost graph library: user guide and reference manual*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[17] Alexandre de Assis Mota, Lia Toledo Moreira Mota, and André Luiz Morelato França. Metodologia orientada a objetos para visualização rápida de grafos não-dirigidos a partir da lista de arestas. In *Anais do XXXVI SBPO - Simpósio Brasileiro de Pesquisa Operacional*, 2004. São João Del Rei, MG.

[18] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[19] P. S. Nagendra Rao and Ravishankar Deekshit. A novel Algorithm for Automatic Generation of One-Line Diagram of Distribution Feeders. *Electrical Power Components and Systems*, 32:1255–1268, 2004.

[20] D. Stewart and Z. Leyk. Meschach. Meschach: Matrix computations in C. `http://www.netlib.org/c/meschach/`.

A Artigo submetido ao WICCGPI 2006

Geração Automática de Diagramas Unifilares

Paulo Meira
Wu Shin-Ting
DCA – Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas
Caixa Postal 6101 – 13083-970 – Campinas
{meira,ting}@dca.fee.unicamp.br

Resumo

Diagramas unifilares constituem uma das mais importantes ferramentas de apoio aos engenheiros de potência tanto para planejamento como para monitoramento de uma rede elétrica. A rápida evolução dos recursos gráficos e das técnicas de visualização assistidas por computador despertou o interesse pelos algoritmos de geração automática de diagramas inteligíveis desde o final da década de 90. Neste trabalho apresentamos uma análise comparativa de três algoritmos relatados na literatura.

1. Introdução

Diagramas unifilares são de extrema importância na análise, no estudo e no planejamento de sistemas elétricos, visto que são a representação mais utilizada [6].

Uma das propriedades essenciais que um diagrama unifilar deve apresentar é uma boa legibilidade. Gerar diagramas unifilares legíveis manualmente através da topologia de uma rede é, no entanto, um processo difícil, trabalhoso e nem sempre os resultados são satisfatórios. Com sistemas elétricos cada vez maiores, essa dificuldade tende a aumentar, assim como a possibilidade de um erro.

O objetivo deste trabalho é o estudo, seguido da implementação, de um conjunto algoritmos de geração automática de diagramas unifilares, observando e analisando suas principais características. O estudo desses algoritmos deve possibilitar uma implementação geral que sirva de apoio à análise de redes de transmissão quanto de distribuição.

O desenho automático de um grafo a partir das informações dos seus nós e dos ramos entre estes nós, sem conhecimento da localização destes, é um assunto que vem sendo estudado desde a década de 1990. Dos trabalhos existentes, selecionamos três deles, ou por sua popularidade como o algoritmo de Fruchterman ou por sua robustez como o algoritmo de Mota e outros, com o intuito de compará-los quanto à legibilidade.

Em 1991, em analogia com sistemas de forças, Fruchterman e outros descrevem uma proposta que reduz o problema de geração de grafo num problema de determinação de uma configuração geométrica de vértices e arestas em equilíbrio onde as distâncias entre os nós são aproximadamente iguais [5]. Este algoritmo é utilizado em várias aplicações diferentes, inclusive na geração automática de diagramas unifilares [7]. No entanto, o posicionamento relativo entre os vértices na configuração final é muito dependente dos seus posicionamentos na configuração inicial.

Em 2004, Mota e outros [4] expõem uma abordagem alternativa do uso de forças para desenho automático de grafos, desta vez elaborada especificamente para o posicionamento dos elementos de um diagrama unifilar. O grande diferencial está na definição das forças entre os nós de tal sorte que sempre resulta numa configuração com mesmo posicionamento relativo entre os vértices, não importando qual seja a configuração inicial destes.

Diagramas unifilares radiais foram explorados por Rao e Deekshit em 2004 [8], aplicando um procedimento que ressalta as características deste tipo de estrutura, como a simetria. Esta proposta deixa de lado o tratamento de conexões cíclicas.

2. Topologia de Redes Elétricas

Para ser autocontido, são apresentadas nesta seção noções básicas sobre redes elétricas e a representação gráfica em diagramas unifilares [6].

Hoje em dia os sistema de energia elétrica são predominantemente trifásicos. Quando um sistema trifásico apresenta fases equilibradas e opera em regime senoidal permanente, os cálculos e a representação do circuito são simplificados através do uso dos modelos chamados unifilares (ou modelo por fase).

O diagrama unifilar é a representação gráfica do modelo unifilar. Se neste diagrama os dispositivos são representados como barras e os condutores elétricos que os interligam como segmentos simples de reta, estamos utilizando o modelo barra-linhas. Este modelo é utilizado neste trabalho, com as barras representadas por quadrados preenchidos. Podemos associar um grafo à topologia de um sistema, utilizando as barras como os nós e as linhas como ramos do grafo.

É importante observar a diferença entre os sistemas elétricos de transmissão e os de distribuição. Estes últimos apresentam, em geral, estruturas em árvore, nas quais surgem ramificações a partir de um alimentador principal. Os sistemas de transmissão apresentam conexões cíclicas para garantir a operação do sistema quando uma linha é interrompida, permitindo que o fluxo de energia se mantenha e o sistema continue em operação.

3. Algoritmo de Rao e Deekshit

3.1. Descrição

O algoritmo consiste em estruturar hierarquicamente os nós de acordo com a distância, medida em termos de número de nós, entre eles. Para isso, é importante a escolha do nó-raiz que usualmente é o alimentador principal. Portanto, é um algoritmo especifico para sistemas de distribuição, visto que não trata conexões cíclicas.

Os passos do algoritmo são:

1. Escolher o nó raiz.
2. A partir do nó raiz, procurar o nó folha mais distante.
3. Posicionar todos os nós entre raiz a esta folha (ramificação principal), sendo a posição de cada nó a posição do seu nó pai somada de um incremento.
4. Para cada um dos nós da ramificação principal, aplicar o seguinte, iniciando do nós mais próximo do nó folha:
   (a) Escolher a orientação da ramificação (esquerda ou direita) – a escolha é arbitrária.
   (b) Aplicar *depth-first search*(DFS) para localizar o nó mais distante [3].
   (c) Posicionar os elementos do ramo que liga o nó da ramificação principal e este nó mais distante. Após estabelecer-se uma posição livre, estes nós são posicionados da mesma forma que os nós da ramificação principal: cada nó-filho é colocado na posição de seu pai somada a um incremento, com exceção do primeiro nó, que é colocado na posição livre estabelecida.
   (d) Repetir o processo a partir de (b), até que todos os nós filhos do nó da ramificação principal estejam posicionados.

3.2. Implementação

Assim como o restante dos algoritmos, a implementação foi feita na linguagem C++, utilizando a biblioteca *Boost Graph Library*(BGL) [2]. Foi utilizada a classe *adjancency_list* para implementar um grafo não-direcionado. A realização do diagrama unifilar fica por parte de estruturas auxiliares que contém outros dados como a posição dos nós, estruturas essas que foram anexadas ao grafo utilizando o conceito de propriedades empacotadas disponibilizada pela BGL.

Para a implementação específica deste algoritmo, é necessária a criação de dois vetores de posições (posicionamento à esquerda e à direita), que são utilizados para controlar as posições já utilizadas, de modo que não haja sobreposição de nós. Estes vetores são indexados através do nível dos nós em relação ao nó raiz, fazendo assim com que todos os nós de um mesmo nível fiquem posicionados sob uma mesma reta. Quando um nó é posicionado em relação ao nó-raiz, a próxima posição livre associada ao seu nível é incrementada.

3.3. Análise dos Resultados

Para a análise, o algoritmo foi testado com diferentes conjuntos de dados, que podem ser carregados de arquivos, como casos teste, ou editados e criados durante a execução do programa implementado.

Apesar da BGL já dispor de uma implementação genérica de DFS, foi implementada uma versão simples que tenta verificar também se há ciclos no grafo, pois o algoritmo não prevê o tratamento de conexões cíclicas.

Apresentamos o resultado da aplicação do algoritmo a dois sistemas. O primeiro, de 90 nós, foi criado para realizar os testes. O segundo, de 14 nós, sistema é um caso teste para fluxo de carga [1], do qual só foram utilizados os dados topológicos.

Nas figuras, o nó mais à esquerda é o nó raiz e a ramificação principal é a que parte dele e se mantém na horizontal. É possível observar claramente quais nós estão no mesmo nivel.

O sistema de 14 nós foi utilizado apenas para ilustrar que o algoritmo não deve ser aplicado a sistemas com conexões cíclicas. Ocorrem várias sobreposições de nós e também de ramos: observe na figura 2 que só é possível contar 9 nós dos 14 totais.

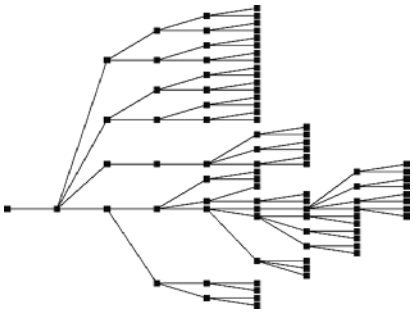

Figura 1. Sistema de 90 nós

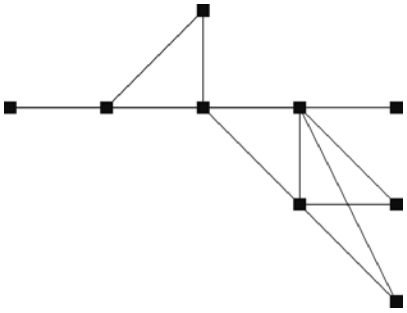

Figura 2. Sistema de 14 nós

4. Algoritmo de Fruchterman

4.1. Descrição

O algoritmo de Fruchterman consiste em determinar, de forma iterativa, uma configuração de equilíbrio entre os vértices de um grafo. O algoritmo funciona como uma anologia a um sistema de partículas que interagem com forças de atração e de repulsão. Neste sistema, as partículas são os nós do diagrama unifilar, as forças de atração são calculadas na direção dos ramos e têm magnitude determinada por seus comprimentos e as forças de repulsão são calculadas entre todos os nós, análogo a uma força eletrostática para cargas de mesmo sinal.

Cada iteração do algoritmo consiste em calcular as forças de atração entre os nós ao longo do ramo que os conectam e as forças de repulsão entre todos os nós simultaneamente. Calculadas as forças resultantes para todos os nós, suas posições são atualizadas. Ao sistema é associada uma temperatura que decresce a cada iteração. Este valor de temperatura é multiplicado pelo incremento de posição antes dele ser somado à posição. Quando a temperatura do sistema chegar num valor próximo do zero, considera-se que ele tenha atingido uma configuração estável na qual não ocorrem mais deslocamentos.

4.2. Implementação

Originalmente, as funções propostas por Fruchterman para determinar os módulos das forças de atração e repulsão são, respectivamente $a(d) = d^2/k$ e $r(d) = k^2/d$, onde d é a distância entre os nós em análise e k é uma constante arbitrária que serve como controle do espaçamento global dos nós. Ao se iniciar a aplicação do algoritmo, o valor da temperatura e e da constante k são inicializados e começam as iterações.

Em cada iteração, são realizados três passos:

1. Determinação da somatória das forças repulsivas entre todos os nós adjacentes.
2. Cálculo das forças atrativas entre cada par de nós.
3. Atualização da posição de cada nó e da temperatura do sistema.

Após cada iteração o algoritmo entra em estado de espera por alguns milissegundos para o usuário visualizar a configuração do grafo e, eventualmente, interromper o processo se o resultado estiver condizente com a sua expectativa antes da temperatura atingir um valor próximo de zero.

4.3. Análise dos Resultados

O algoritmo foi aplicado aos dois sistemas utilizados para o algoritmo de Rao e Deekshit, além do sistema de 57 barras disponibilizado com caso teste para fluxo de carga [1].

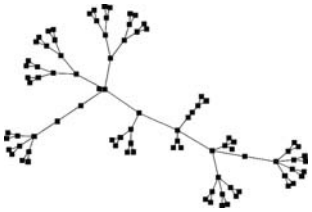

Figura 3. Fruchterman aplicado ao sistema de 90 nós

O sistema de 90 nós, sem conexões cíclicas, apresenta uma forma orgânica. Observa-se claramente uma ramificação principal, mais longa, a partir da qual surgem as ramificações menores.

Para os dois outros sistemas, observa-se que o comprimento das arestas não é aproximadamente igual como o esperado. Existem ramos com o dobro de comprimento de outros. A legibilidade é prejudicada pela alta concentração de nós no canto à esquerda do diagrama.

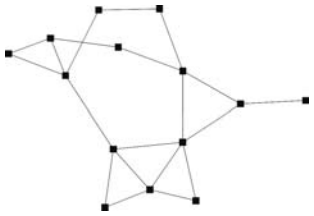

Figura 4. Fruchterman aplicado ao sistema de 14 nós

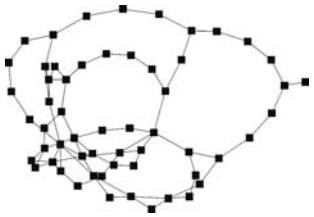

Figura 5. Fruchterman aplicado ao sistema de 57 nós

5. Algoritmo de Mota

5.1. Descrição

Este algoritmo também é baseado em analogia com um sistema de partículas sujeitas a um conjunto de forças, mas apresenta algumas diferenças essenciais. Ao contrário do algoritmo de Fruchterman, não é necessário aplicar o conceito de temperatura para que o sistema entre em equilíbrio. O algoritmo sempre converge a uma configuração estável após um certo número de iterações, sem precisar recorrer ao uso do parâmetro de temperatura como um critério de parada.

A força nos ramos pode ser tanto atrativa quanto repulsiva, sendo análoga à força de Hooke (molas): $f_r(d) = K_r(l-d)/3d$, onde d é a distância entre os nós em análise, l o comprimento natural da aresta e K_r uma constante de ajuste. A força entre os nós é de repulsão, sendo dada por $f_n(D) = K_n/D$, mas é calculada somente uma vez por nó, pois D é a soma dos deslocamentos entre um nó e os restantes. A constantes K_r e K_n servem para ajustar o espaço ocupado pelo diagrama final. O algoritmo prevê também o tratamento de nós fixos, nos quais não são aplicadas forças.

5.2. Implementação

Na implementação foi considerado que todas as arestas tem o mesmo comprimento natural, igual a 1. A implementação foi muito próxima à do algoritmo de Fruchterman, com as diferenças já exploradas na descrição.

5.3. Análise dos Resultados

O algoritmo de Mota foi aplicado aos mesmos três sistemas para os quais aplicamos o algoritmo de Fruchterman anteriormente.

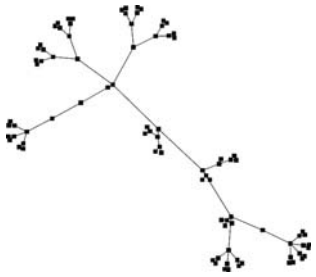

Figura 6. Mota aplicado ao sistema de 90 nós em árvore

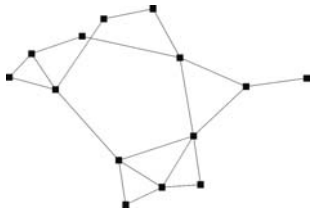

Figura 7. Mota aplicado ao sistema de 14 nós

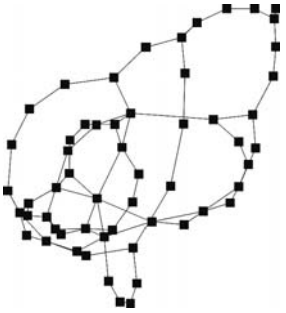

Figura 8. Mota aplicado ao sistema de 57 nós

Para o sistemas de 90 nós, os ramos que terminam em nós folhas têm seus comprimentos bastante reduzidos. Esse é um comportamento geral deste algoritmo: quando mais próximo da extremidade, menor o comprimento de um ramo.

Aplicado aos sistemas com ciclos, foi observada uma boa distribuição de comprimentos de ramos. Esta característica é favorecida para sistemas com grande número de ciclos e é prejudicada quando há muitas folhas.

6. Conclusões e Trabalhos Futuros

Comparando os resultados dos algoritmos, é notado que, quando o grafo apresenta uma hierarquia e não apresenta ciclos, o resultado do algoritmo de Rao e Deekshit é satisfatório, pois permite a análise rápida de um nó com relação a seu nível e facilmente traça-se o percurso entre um nó e o nó raiz.

Quando comparamos a aplicação dos algoritmos baseados em força ao sistema sem ciclos, observamos que o algoritmo de Fruchterman apresenta uma estrutura mais agradável e regular. Já quando comparamos o desempenho destes algoritmos quando aplicados a sistemas com conexões cíclicas, o algoritmo de Mota resulta em comprimentos de ramos mais regulares e uma distribuição mais uniforme de nós.

Visto que cada algoritmo é capaz de trazer alguma característica desejável ao diagrama unifilar gerado, propomos futuramente implementar algoritmos híbridos como, por exemplo, aplicar o algoritmo de distribuição de Rao e Deekshit para posicionar somente a ramificação principal, deixando o posicionamento do restante dos elementos para o algoritmo de Mota.

Para os dois algoritmos baseados em força, é comum que dois nós sejam posicionados muito próximos. Uma restrição de distância mínima não baseada em forças poderia ser utilizada para adequar o posicionamento. Em algumas situações, resultados melhores são obtidos ao se aplicar alternadamente os algoritmos, chaveando após algumas iterações.

Em trabalhos futuros, pretendemos também avaliar os desempenhos dos algoritmos quando são aplicados a somente uma parte dos nós de um grafo, mantendo o restante fixo. Esta situação ocorre, por exemplo, quando já são conhecidas as posições dos nós e deseja-se substituir parte deles por sistemas equivalentes, que ainda não terão posições associadas.

Referências

[1] Power systems test case archive. http://www.ee.washington.edu/research/pstca/.

[2] *The boost graph library: user guide and reference manual*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, second edition edition, 2001.

[4] A. de Assis Mota, L. T. M. Mota, and A. L. M. França. Metodologia orientada a objetos para visualização rápida de grafos não-dirigidos a partir da lista de arestas. In *Anais do XXXVI SBPO - Simpósio Brasileiro de Pesquisa Operacional*, 2004. São João Del Rei, MG.

[5] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[6] A. Monticelli and A. Garcia. *Introdução a sistemas de energia elétrica*. Editora Unicamp, 2003.

[7] Y. Ong, H. Gooi, and C. K. Chan. Algorithms for automatic generation of one-line diagrams. *IEE Proceedings-, Generation, Transmission and Distribution*, 147:292–298, 2000.

[8] P. S. N. Rao and R. Deekshit. A novel algorithm for automatic generation of one-line diagram of distribution feeders. *Electrical Power Components and Systems*, 32:1255–1268, 2004.