Diagrama de Voronoi Distribuído

Frederico Martins Biber Sampaio

***Resumo*—Trabalho prático sobre implementação de um modelo de diagrama de Voronoi distribuído.**

***Keywords*—Algoritmo distribuído, diagrama de Voronoi, triangulação de Delaunay, simulador.**

I. Introdução

Este trabalho prático apresenta uma fundamentação, especificação e solução do problema de elaboração de um algoritmo distribuído para o diagrama de Voronoi. Uma implementação da solução do problema foi elaborada por meio de um simulador para algoritmo distribuído, o DAJ, e um módulo de visualização dos diagramas de Voronoi de cada processo.

II. Fundamentos

A. Diagrama de Voronoi

O diagrama de Voronoi é caracterizado por locais mais próximos de um determinado ponto. Diversas referências relacionam sua origem ao problema de se estabelecer regiões postais [1]. Devido à sua natureza, o problema é equivalente à minimização de distâncias. Por isso, o diagrama de Voronoi tem aplicação prática na otimização de muitos tipos de sistemas como transportes, comunicação, marketing, estudo de áreas de influência, etc. Considerando-se a Ciência da Computação, sua área de estudo está relacionada com computação geométrica.

Cada ponto de um diagrama de Voronoi é chamado de site. As regiões formadas pelos sites são chamadas de células de Voronoi. Para um diagrama com apenas um site, sua célula ocupa todo espaço disponível. Para mais de um site, aplica-se a regra geral e fundamental:

$$Vor(p, P) = \{x : |px| \leq |qx|, \forall q \in P\} \quad (1)$$

Os locais com distância igual entre dois sites formam arestas de Voronoi. A interceção das arestas formam os vértices de Voronoi. Os vértices são locais onde mais de dois sites possuem a mesma distância. Podem existir um número $n > 2$ indeterminado de sites equidistantes de um mesmo vértice de Voronoi. Porém, esse e outros casos exemplificam limtes do modelo adotado neste trabalho para a soluçao do problema.

As arestas de Voronoi podem não ser limitadas por outras arestas (formando vértices). Note-se o caso de existirem apenas dois sites. A única aresta é uma reta sem limitação em ambas as direções. Também não é possível formar vértice pois o número n de sites não é suficiente.

B. Triangulação de Delaunay

O problema de Voronoi tem um dual que é a triangulação de Delaunay. Todo e qualquer triângulo forma um círculo que tange seus vértices; o circuncírculo do triângulo. Na triangulação de Delaunay, um conjunto de pontos no espaço formam triângulos desde que não existam pontos no interior de seus circuncírculos. Novamente, é possível que infinitos pontos ocorram no mesmo circuncírculo.

No caso, os pontos de Delaunay são equivalentes aos sites de Voronoi. Os circuncentros dos triângulos de Delaunay formam os vértices do diagrama de Voronoi. Contudo, por causa do fundamento geométrico básico, o triângulo, algoritmos de triangulação são limitados aos casos no quais cada circuncírculo possua apenas e exatamente os pontos de um único triângulo. Em outras palavras, o número de sites mutuamente equidistantes no diagrama de Voronoi pode estar limitado a três. Em alguns algoritmos, mesmo que essa limitação possa tornar incertos os resultados (não determinístico), é possível aceitar mais de três pontos pertencentes a um circuncírculo, desde que apenas três representem um triângulo na triangulação de Delaunay.

As arestas de Voronoi, ao menos as que são limitadas, podem ser definidas pela sequência de vértices, obtidas pelos circuncentros da triangulação. Porém, as arestas que não são limitadas não podem ser definidas pela triângulação de Delaunay. Note-se que todas as arestas de Voronoi não limitadas irão se originar em sites pertencentes ao *convex hull* [2].

Para simplificar a solução do problema, será adotada uma estratégia de usar um triângulo inicial que atuará como um *convex hull*. Assim, a

triângulação ocorrerá sempre no interior do *convex hull*. Ou seja, todos os sites devem estar na área interior do triângulo inicial, que deve ser suficientemente grande [1]. Assim, o problema de Voronoi se iguala conceitualmente e operacionalmente ao problema de Delaunay, pois sempre haverá ao menos um triângulo no diagrama, fornecendo as condições mínimas para a triangulação. Além disso, as arestas de Voronoi sempre serão delimitadas pelos triângulos exteriores, baseados nos vértices do triângulo inicial. Esta estratégia é adotada pelo software de referência adotado como base para a elaboração do presente trabalho [3].

Os problemas de Delaunay e de Voronoi podem ser solucinados por operações vetorias em espaços euclidianos de múltiplas dimensões. Para simplificar a simulação gráfica do problema, o modelo adotado neste trabalho prático também limita o espaço em duas dimensões, ou seja, ao plano cartesiano.

Outra limitação importante do modelo de Voronoi e Delaunay é que todo ponto (site) deve ter posição única e diferente no espaço. Todos os algoritmos pesquisados dependem dessa limitação e algumas implementações fazem sua verificação em forma de pré-condição.

C. Ambiente e especificações do algoritmo distribuído

No modelo de sistemas distribuídos, diferentes processos se interagem por meio de envio/recebimento de mensagens para atingir um objetivo em comum ou global. Os objetivos individuais ou as funções dos processos podem ser diferentes. De qualquer forma, cada processo conhece apenas seu estado local e, usando apenas o mecanismo de troca de mensagens, eles podem se comunicar com outros processos.

É necessário definir um modelo que relacione o algoritmo distribuído com o diagrama de Voronoi. No modelo adotado neste trabalho, cada site representará um processo. Todos os processos possuem o mesmo objetivo, as mesmas funções e executam o mesmo algoritmo. O objetivo de um site é conhecer o diagrama de Voronoi formado pelos seus vizinhos (no espaço). Um site precisa conhecer ao menos os sites que sejam considerados relevantes. O modelo pode adotar diferentes critério de relevância.

Como o diagrama de Voronoi modela o espaço, é fundamental que cada processo conheça sua posição. Como sistemas distribuídos relacionados com espaço normalmente também são relacionados com redes móveis, os processos devem ser capazes de se moverem. Assim, o diagrama de Voronoi também deve ser atualizado dinamicamente, por meio de operações básicas de adição, movimentação e remoção de sites.

O modelo deve ser independente de (1) topologia de conexão dos processos, (2) sincronização ou (3) modelo de falhas. A condição mínima pressuposta é que os processos formam uma rede conectada. Além disso, se um site remoto é relevante, ele deve ser atingível passando por um número máximo predefinido de links (profundidade ou saltos). Os processos devem possuir um estado local válido e, em tempo indeterminado, seu estado deve ser capaz de representar um diagrama de Voronoi equivalente à disposição geométrica "efetiva" (ou "real") da localizão dos sites, ao menos em relação aos sites remotos relevantes.

III. SOLUÇÃO

O software para a solução do problema é baseado na seguinte arquitetura:

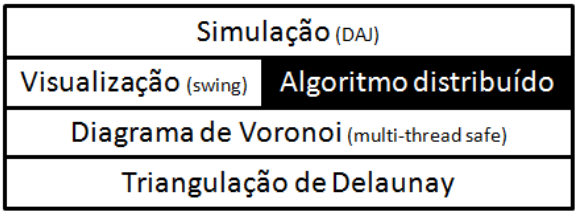

Figura 1: Arquitetura geral do software

IV. ALGORITMOS BÁSICOS

Para possibilitar o funcionamento do modelo proposto para o algoritmo distribuído, o primeiro passo é elaborar um software capaz de criar um ambiente

[1] Por causa de operações quadráticas (exponenciação e radiciação), a precisão dos cálculo de ponto flutuante pode impor um limite prático para o tamanho do triângulo inicial. Falhas de precisão podem impossibilitar a conclusão correta das triangulações, levando a diagramas incorretos e falhas na execução do programa. Algumas técnicas eficientes de ajuste de precisão podem ser usadas, principalmente nos casos de comparação com zero. Porém algumas imprecisões são difícieis ou impossíveis de se tratar, como é o caso das comparações entre valores muito diferentes, como no caso das distâncias. Em resumo, operações de ponto flutuante podem representar outra origem de limitação da presente solução computacional do problema.

de controle de diversos diagramas de Voronoi, um para cada processo local. Considerando a dinâmica de inclusão, remoção e movimentação, o algoritmo do diagrama de Voronoi deve ser capaz de adicionar e remover sites. A movimentação é uma operação discreta e pode ser simulada por uma remoção da antiga posição seguida da inclusão na nova posição.

A simulação utiliza o software DAJ [4]. O simulador possui alguns cenários pré-estabelecidos que podem ser selecionados e configurados no início da execução.

O ideal é que os diagramas possam ser visualizados de forma dinâmica, ao longo da execução da simulação. Por isso, o algoritmo de visualização executa em paralelo ao algoritmo distribuído, lendo os dados do Diagrama de Voronoi e da Triangulação de Delaunay. Para evitar que a vizualização leia dados que podem ser alterados simultaneamente pelo algoritmo distribuído, a classe de diagrama de Voronoi deve permitir bloqueios (*lock*) entre as operações de visualização e as eventuais alterações do diagrama causados pelo algoritmo distribuído.

Depois de uma pesquisa inicial, optou-se por adotar um software já existente como base de referência para o presente trabalho [3]. O software base usa triangulação de Delaunay. Os algoritmos de triangulação adotam espaços de n-dimensões. Porém, os programas de visualização adotam apenas duas dimensões. A escolha de adaptar um software existente ajudou na fase inicial, mas acabou não sendo uma opção completa. A análise e entendimento do software foi dificultada por causa da complexidade algébrica das operações sobre número arbitrário de dimensões. O programa de visualização foi reescrito para possibilitar controle e seleção de múltiplos diagramas, além de funcionar em paralelo com o simulador. Mais importante, o software não possuía recurso de remoção de site de Voronoi, que é equivalente a remover um ponto da triangulação de Delaunay.

É possível utilizar diferentes estruturas de dados para manter as informações da triangulação de Delaunay. O software base adotou um controle de triângulos vizinhos. Os vizinhos são triângulos que compartilham um lado (face) em comum. Para isso, existe uma coleção com todos os triângulos e, para cada triângulo, existe outra coleção com os triângulos vizinhos. Apesar de ser uma estrutura de dados mais complexa de ser mantida ao longo das operações de inclusão e remoção, ela tende a ser mais eficientes que outras mais simples [5].

Obviamente, a lista de vizinhos de cada triângulo possui, no máximo, três elementos. Os triângulos externos, relacionados com o triângulo inicial, possuem menos de três vizinhos. A ordem, no sentido horário ou anti-horário, (1) de busca, (2) de disposição dos vizinhos e (3) da disposção dos vértices dos triângulos é fundamental para o correto funcionamento dos algoritmos de triangulação. Mesmo quando a ordem não é imposta nas estruturas de dados, ela deve ser considerada ao longo da execução para a realização dos cálculos.

A adição e a remoção de site do diagrama são operações que podem ser consideradas “inversas”, pois a remoção busca desfazer o o resultado da inclusão. No geral, ambas são operações que lidam com cavidades geradas pelos sites [6].

Na inclusão, a cavidade é formada pelo conjunto de triângulos afetados (invalidados) pelo novo site. Esse conjunto é composto por todos os triângulos que circunscrevem o novo site. Os triângulos desse conjunto devem ser removidos, mantendo-se apenas os lados opostos ao novo site para formar uma cavidade. A nova triangulação é simples, pois basta conectar os pontos ao longo da cavidade diretamente com o novo site, formando novos triângulos, todos com o novo site como vértice em comum.

Na remoção, a cavidade é obtida removendo-se os triângulos com vértice no site a ser removido, operação exatamente inversa ao que ocorre na inclusão. O processo mais complexo é definir a nova triangulação dentro da cavidade. No geral, combina-se os pontos da cavidade buscando triângulos válidos. Um triângulo é válido se ele forma uma “orelha” que “escuta” o site removido. Mais especificamente, a direção dos pontos, no sentido horário ou anti-horário, do triângulo candidato deve ser a mesma da direção de um triângulo cujo site removido substitui um dos vértices do triangulo candidato. Isso evita que qualquer triangulação ocorra do lado de fora da cavidade. Se o triângulo é totalmente interno à cavidade, ele deve obedecer ao princípio básico de Delaunay: seu circuncírculo não pode conter qualquer outro ponto (site) no seu interior. A cada triângulo válido encontrado, a cavidade se reduz, até sobrarem apenas três pontos (o mínimo para um triângulo). A complexidade de tempo da busca de triângulos válidos pode ser reduzida utilizando-se listas de prioridade [6].

Uma das principais tarefas das operações de in-

clusão e remoção é, ao final, manter correto o controle de triângulos vizinhos. No caso da remoção, apesar do ajuste do controle de vizinhos ser relativamente simples e direto, ele não foi encontrado em nenhuma referência pesquisada sobre triangulação de Delaunay.

Outro ponto importante é a complexidade de tempo da busca de pontos no espaço da triangulação. As buscas adotadas no software base [3], partem do último triângulo adicionado na triangulação, percorrendo seus vizinhos. Considerando-se que os sites sejam adicionados ou removidos por proximidade, essa técnica tem grande potencial prático de otimização. Se o ponto pesquisado não se relaciona com o triângulo mais recente ou seus vizinhos, a busca é reiniciada de forma linear, ou seja, passando sequencialmente por todos os triângulos até encontrar uma correspondência. Parece que a busca, como tem característica espacial, pode ser acelerada com técnicas de divisão do espaço como, por exemplo, as Quadtree. Contudo, essa técnica não foi experimentada no presente trabalho prático, ficando a sugestão para a eventualidade de uma versão futura.

A implementação dos diagramas foi dividida nas seguintes classes da linguagem Java:

- Pnt: classe que representa os pontos no espaço n-dimensional e implementa os principais cálculos necessários para a triangulação.
- ArraySet: coleção usada para a estrutura de triângulos vizinhos.
- Graph: implementação da estrutura de dados de busca de triângulos e vizinhos.
- Triangle: classe que define os objetos triângulos.
- DelaunayTriangulation: classe que define os objetos que fazem a triangulação de cada site, incluindo as operações de inclusão e remoção.

A implementação da visualização dos diagramas foi dividida nas seguintes classes da linguagem Java:

- DelaunayAp: classe que fornece a interface gráfica para o desenho e seleção dos diagramas de Voronoi, com opção para visualização da triangulação de Delaunay e dos circuncírculos.
- DelaunayPanel: classe que faz o desenho efetivo de cada diagrama.
- VisualVoronoiDiagram: classe que define os objetos de controle das operações sobre o diagrama de Voronoi de forma paralela (*multi-thread*), em especial os travamentos () que separam as operações concorrentes de desenho (readonly) e de atualizações (read/write).
- VisualVoronoiDiagramFactory: classe que define os construtores da classe VisualVoronoiDiagram[2].

V. ALGORITMO DISTRIBUÍDO

Por causa do fator espaço e localização, no modelo distribuído, os canais de ligação (*links*) entre os processos tende a corresponder ao modelo de comunicação sem fio, caracterizado por regiões de alcance de sinais eletromagnéticos e comunicação via *broadcasting*. Contudo, o simulador adotado, o DAJ, utiliza um modelo de comunicação de links direcionais fixos. Assim, infelizmente, o simulador impõe uma limitação e contribui para a definição das características do algoritmo adotado.

O modelo adotado é dinâmico, ou seja, considera a movimentação dos sites em uma área plana definida e limitada tanto pelo simulador quanto pelo triângulo inicial que representa um *convex hull*. Todavia, considerada a limitação do simulador, os links não são refeitos ou ajustados para refletir a proximidade espacial.

O modelo adota três tipos de mensagens:

- Presença: notifica a presença de um nó na rede.
- Ausência: notifica a intenção de um nó em sair da rede.
- Movimento: notifica a mudança de localização de um nó da rede.

Na prática, a mensagem de movimentação não seria necessária, pois bastaria que o site remoto fizesse um anúncio de sua presença em uma nova posição, uma vez que o site local conhece a posição anterior, registrada no diagrama. Essa mensagem foi mantida para ampliar a semântica do modelo de comunicação.

Todo processo possui uma lista com a última mensagem recebida de cada processo remoto. Por isso, cada mensagem enviada deve ter uma identificação relativa ao índice do estado do processo remoto que originou a mensagem. O índice do estado de um processo é um número crescente,

[2] O uso do *design pattern factory* generaliza o uso do algoritmo distribuído para aplicações com ou sem interface gráfica de simulação ou execução paralela. O *factory* permite usar objetos de diferentes subclasses, sem conhecê-las previamente.

adicionado a cada mudança relevante do estado interno, em especial antes do envio e logo após a recepção de mensagem.

Considerando a necessidade de se distinguir os processos no histórico e saber sua posição no espaço, cada mensagem também possui uma identificação única do processo originário e sua localização no momento do envio. A identificação única de cada processo é outra característica do modelo (limitação).

Para evitar que se propague indefinidamente, cada mensagem também possui um indicador de "profundidade" em forma de contador decrescente. Na recepção, o processo subtrai o contador de profundidade da mensagem. Se o valor for maior que zero, ele repassa a mensagem original com contador reduzido para os demais links (exceto o link de entrada da mensagem).

No modelo, os processos se comunicam por meio de dois mecanismos de transmissão de dados: (1) *flooding* de mensagens de *broadcast* ao longo da rede, quando o destino da mensagem não é definido, e (2) *forward* de mesnagem de *unicast* ao longo de um caminho (*path*) distribuído, no caso do processo local conhecer um link que se cominica com o processo remoto de destino.

Caso o processo local receba uma mensagem com destino (*unicast*), mas não conheça um caminho para ele, a mensagem pode ser ignorada ou transmitida por alagamento. Independente do processo local ser o destino de uma mensagem, seu conteúdo pode ser usado para atualizar o estado local, caso o site remoto originário seja relevante e a mensagem seja mais atual do que a última registrada no hitórico local.

Para permitir o encaminhamento (*forward*), o histórico local mantém a informação do canal (*link*) que recebeu mensagens originadas de cada site remoto. Para tentar otimizar o caminho até o destino, o hitórico mantém o canal cuja mensagem de destinho atravessou menos processos (*hops*). Por isso, cada mensagem também carrega a informação da profundidade (*deep*) originalmente informada pelo processo originário.

O histórico local pode ter tamanho (em memória) limitado. Baseado na última mensagem em histórico, é possível realizar o controle de posicionamentos dos nós remotos da rede que são atingíveis pela comunicação por alagamento. Porém, supondo que a comunicação possa ter falha, a posição é controlada na classe de diagrama distribuído, e não no histórico.

De qualquer forma, o histórico ajuda a reduzir o alagamento, pois possibilita a identificação de *loop*. O *loop* também pode ocorrer no encaminhamento, caso o caminho distribuído forme laço. Para evitar propagação de mensagem em *loop*, caso um processo local receba uma mensagem originada de um processo remoto específico e seu índice de estado seja menor ou igual ao da última mensagem registrada do mesmo site remoto no histórico local, certamente ela já foi anteriormente recebida e, possivelmente, também já foi transmitida. Por isso, a mensagem "antiga" pode ser ignorada.

No geral, um processo deve conhecer ao menos sua célula de Voronoi. Assim, cada processo deve manter no diagrama, ou mesmo no histórico, ao menos seus vizinhos (geométricos) que afetam (ou determinam) sua célula de Voronoi. Este é um critério de relevância mínimo considerando o problema do trabalho. Contudo, como já mencionado, manter mais sites em histórico ajuda na redução da comunicação por alagamento. Também é possível adotar outros critérios de relevância, que incluam mais sites do que este critério mínimo.

Um movimento causa uma mudança de posição detectada no site local e, certamente, causa mudança na forma da sua célula de Voronoi. Essa mudança também afeta seus vizinhos e, por isso, deve ser anunciada na rede. Como os links não são diretamente relacionados com a posição dos sites no espaço, mesmo não sendo afetado por uma inclusão, remoção ou movimentação, um processo que recebe uma mensagem deve fazer seu repasse, sempre considerando o controle de profundidade.

A mensagem de "movimentação" deve ser suficiente para ser tratada como "presença" caso o processo remoto não seja conhecido pelo processo local (receptor). Neste caso, o processo local também pode se anunciar por meio do envio de uma mensagem de "presença" destinada ao novo site remoto, pois o processo remoto também pode não ter histórico sobre ele. Além disso, uma nova mensagem de presença pode ser emitida em intervalos de tempos (aleatórios ou regulares), como forma de atualização dos processos remotos. Os processos podem optar por remover sites remotos que não anunciam sua presença por longo período de tempo. A renotificação também possibilita o tratamento de casos como falhas na comunicação, mudança na to-

pologia da rede (invisível ao diagrama), movimento para áreas desconhecidas (sem histórico registrado), etc. Esses reenvios representam uma espécie de mecanismo de estabilização, pois possibilita uma convergência para estados locais mais precisos (mais próximos da "realidade").

Além de ignorar mensagens mais "antigas", verificando o índice do estado do processo remoto em histórico, o processo local também pode não tratar mensagens de movimentação ou presença sem mudança de local conhecido e saída (ausência) de nós desconhecidos. Mesmo não sendo necessário o tratamento, qualquer mensagem mais "recente" deve sempre ser registrada em histórico.

Nos casos que envolvam inclusão[3] de site remoto no diagrama, o site local pode definir a sua relevância. Porém, na sua própria movimentação, o site local pode não ser capaz de avaliar a relevância dos sites remotos, pois eles podem sequer serem conhecidos ou já terem sido eliminados do histórico. Vários sites podem se tornar irrelevantes ao longo da movimentação, pois ficam distantes, encobertos ou, simplesmente, não afetam a célula de Voronoi do processo local. Por isso, independente do critério de relevância ou do controle desse critério, de tempos em tempos, cada nó pode reavaliar seu diagrama e eliminar os sites irrelevantes. O site pode não estar presente no diagrama, mesmo assim manter um histórico local. Ou seja, a remoção certamente retira o site remoto do diagrama local, mas não necessariamente remove do hitórico local. Esse tipo de remoção baseada em critério de relevância também pode representar um mecanismo de otimização do uso dos recursos, em especial da memória.

Outro recurso especificado no modelo distribuído é a análise de relevância de sites remotos. O critério de relevância pode variar. Por exemplo, um site remoto pode ser relevante se sua distância até o site local for menor que um raio de cobertura. O critério adotado para o presente trabalho foi: um site remoto é relevante se afetar a célula de Voronoi do site local. Os sites irrelevantes podem ser removidos tanto do diagrama quanto do histórico[4]. A análise da relevância de um site remoto ocorre de duas maneiras: 1) Forma dinâmica, antes da inclusão, que também ocorre no final da movimentação. 2) Forma estática, a qualquer momento o conjunto de sites do diagrama é verificado e os sites remotos irrelevantes são removidos. A análise estática pode ocorrer em períodos de tempo fixos ou variáveis. Transmissões em tempo aleatório podem reduzir tanto eventuais colisões quanto momentos de "pico" de tráfego.

Por fim, o modelo e os algoritmos de triangulação não admitem sites com posições iguais. Porém, na prática, devido a limitações do simulador ou da eventual precisão dos sensores de localização, é possível que dois ou mais sites possuam a mesma posição no diagrama de Voronoi. Um site remoto em colisão poderia ser ignorado, na esperança de que, no futuro, uma nova mensagem de movimentação ou presença em local diferente seja recebida e, não ocorrendo novamente colisão, o site possa ser adicionado, se for relevante. Contudo, isso causa uma invitável perda de controle, principalmente na movimentação do site, pois a operação de remoção pode excluir sites relevantes que podem demorar a "retornar"pelos mecanismos de estabilização.

O software elaborado para o presente trabalho prático lida com essa limitação fazendo um controle de sites por localização. Os sites são agrupados por localização, possibilitando identificação e controle das colisões. Na prática, um site no diagrama se torna um conjunto de um ou mais processos (local ou remoto). Se, ao incluir um site remoto, sua localização não possuir outros sites, o site deve ser adicionado ao diagrama. Se, ao remover um site remoto, sua localização não possuir outros sites, o site deve ser removido do diagrama.

Todos estes algoritmos foram implementados em forma de classes da linguagem Java adaptadas para o uso no simulador DAJ:

- Main: classe que ativa a aplicação do simulador DAJ, fornecendo os casos para as simulações (testes).
- Location2D: classe simples de localização do site no plano (x,y).
- Site: classe de objetos que representam os sites de Voronoi, que informa sua identificação e posição.
- Msg: classe que modela os objetos de mensagens.
- Prog: classe que implementa o comportamento de cada processo (site local), em especial o tratamento de mensagens.
- VoronoiDiagram: classe que define cada diagrama de Voronoi e faz o controle de

[3] Tanto a "presença" quanto a "movimentação" envolvem operação de inclusão.

[4] O diagrama e o histórico possuem funções diferentes. Um site pode não estar no diagrama, mas estar no histórico, e vice-versa.

localização com múltiplos sites.

- DistributedVoronoi: classe que define o objeto de controle do diagrama de Voronoi de cada processo, em especial a adição, movimentação e remoção de sites remotos.
- LocalHistory: classe que define o objeto de controle de histórico dos processos remotos.
- TimerControl: classe que define os objetos que controlam a execução dos eventos periódicos ou com atraso (delay).
- VoronoiDiagramFactory: classe (*design pattern*) que define objetos que generalizam a construção de objetos da classe VoronoiDiagram e suas descendentes.

As demais classes do trabalho são do simulador DAJ ou pertencem aos recursos padronizados da linguagem Java.

VI. Conclusão

O trabalho apresenta um problema clássico em computação geométrica, com várias aplicações relevantes em computação espacial e distribuída. Temas como triangulação de Delaunay também possuem ramificações em outras áreas computacionais, em especial a computação gráfica. Por isso, o problema em estudo tem grande potencial de "otimização" com uso de técnicas como computação paralela e aceleração por hardware via GPU. Dezenas de artigos pesquisados tratam sobre temas relacionados.

Como de praxe, o volume e o tempo de pesquisa e elaboração do software foi consideravelmente maior que o esperado. Muito esforço foi investido na parte do algoritmo geométrico (triangulação de Delaunay), tanto na dinâmica do modelo (inclusão e remoção de site) quanto nos ajustes no *applet* de visualização [3]. Além disso, um grande esforço foi empregado para um controle mais detalhado do algoritmo distribuído relacionado com o diagrama de Voronoi.

O trabalho ainda possui muitos pontos e temas que podem ser melhorados. A principal ausência foi a análise mais detalhada da complexidade de tempo dos algoritmos elaborados. Também não houve nenhuma análise formal sobre os resultados como, por exemplo, a influência das diversas opções de configuração em diferentes cenários, em especial redes com diferentes topologias e tamanhos. Alguns recursos, apesar de terem sido testados, não foram automatizados no simulador. Um exemplo é a movimentação que poderia ter padrões e/ou cenários gerados automaticamente pelo simulador. Por fim, como o trabalho foi o primeiro do curso de Algoritmos Distribuídos, muitas das teorias, modelos e outros conhecimentos adquiridos ao longo do curso poderiam ter sido usados tanto na fundamentação quando na elaboração do software. De qualquer forma, estes pontos parecem ir além do objetivo didático proposto inicialmente para o presente trabalho. Porém, academicamente é importante destacá-los para a eventual continudade e/ou pesquisas futuras.

O trabalho forneceu a oportunidade para atividades práticas como a identificação de problemas na comunicação e estabilização dos estados, a caracterização e estudo das técnicas de distribuição das mensagens, assim como diversos outras questões relacionadas com o processo de elaboração e implementação de algoritmos distribuídos. Certamente, este trabalho prático colocou em perspectiva mais concreta e reforçou a importância de vários dos conceitos relacionados com Algoritmos Distribuídos abordados em sala de aula e/ou nos diversos materiais de estudo.

Referências

[1] M. de Berg, *Computational Geometry: Algorithms and Applications*. Springer, 1997. [Online]. Available: http://books.google.com.br/books?id=_vAxRFQcNA8C

[2] F. Aurenhammer and R. Klein, *Voronoi Diagrams*, ser. Informatik-Berichte. Karl-Franzens-Univ. Graz & Techn. Univ. Graz, 1996. [Online]. Available: http://books.google.com.br/books?id=27vkSgAACAAJ

[3] L. P. Chew. (2007, Dec.) Voronoi/delaunay applet. [Online]. Available: http://www.cs.cornell.edu/Info/People/chew/Delaunay.html

[4] W. Schreiner. Daj - a toolkit for the simulation of distributed algorithms in java.

[5] M. Gahegan and I. Lee, "Data structures and algorithms to support interactive spatial analysis using dynamic voronoi diagrams," *Computers, Environment and Urban Systems*, vol. 24, no. 6, pp. 509 – 537, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0198971500000077

[6] M. A. Mostafavi, C. Gold, and M. Dakowicz, "Delete and insert operations in voronoi/delaunay methods and applications," *Comput. Geosci.*, vol. 29, no. 4, pp. 523–530, May 2003. [Online]. Available: http://dx.doi.org/10.1016/S0098-3004(03)00017-7