IMPLEMENTAÇÃO DISTRIBUÍDA DO ALGORITMO DE DIJKSTRA ATRAVÉS DE SISTEMAS MULTIAGENTES

Filipe de Oliveira Saraiva, Aldir Silva Sousa, Eduardo Nobuhiro Asada

Laboratório de Análise de Sistemas de Energia Elétrica
Escola de Engenharia de São Carlos
Universidade de São Paulo
São Carlos, São Paulo, Brasil

Emails: filipe.saraiva@usp.br, aldirss@usp.br, easada@usp.br

Abstract— The Dijkstra algorithm is a classical algorithm which, for a given graph, calculates the shortest path between one vertex and all others vertexes of the graph. This algorithm has important application to areas such as path determination problems, production chain studies and electrical transmission grids restoration. Since the development of distributed computing, it has opened new possibilities concerning to the development of systems to calculate and monitor the status of a model in a descentralized way. We propose the application of multi-agent systems in order to take advantage of theses features and, in a collaborative manner, to perform the Dijkstra algorithm through distributed computation and ?nding the solution of the proposed problem.

Keywords— Dijkstra Algorithm, Distributed Artificial Intelligence, Multi-agent Systems

Resumo— O algoritmo de Dijkstra é um clássico algoritmo computacional onde, dado um grafo, calcula-se o caminho mínimo entre um vértice específico e todos os demais vértices que formam o grafo. Este algoritmo tem importantes aplicações em áreas tão distintas quanto problemas de determinação de caminhos, estudos de uma cadeia de produção ou a restauração de sistemas elétricos de transmissão. Com o desenvolvimento da computação distribuída, novas possibilidades se abrem para o desenvolvimento de sistemas que permitam o cálculo e o acompanhamento do estado do modelo de forma descentralizada. Na busca de assumir estas características e usufruir de suas vantagens, propomos o uso de sistemas multiagentes para que estes, de forma colaborativa, executem o cálculo distribuído do algoritmo de Dijkstra e encontrem a solução para o problema proposto.

Palavras-chave— Algoritmo de Dijkstra, Inteligência Artificial Distribuída, Sistemas Multiagentes

1 Introdução

Muitos problemas enfrentados no planejamento e operação de empresas e organizações podem ser modelados na forma de um grafo para que, em seguida, a obtenção da solução seja dada através da resolução de um problema de caminho mínimo. Alguns exemplos podem ser encontrados em (Arkin and Silverberg, 1987), (Sghaier et al., 2010) e (Hu et al., 2009).

Existem alguns problemas desse tipo cuja resolução pode se beneficiar dos avanços atuais da computação distribuída, onde tanto o processamento quanto a própria obtenção dos dados para o cálculo se dá em um ambiente de computação descentralizada, disperso geograficamente. Os atuais ambientes de computação modernos são distribuídos e de grande porte.

Estes ambientes necessitam de algoritmos adaptados a suas características. Somente assim será possível usufruir as vantagens dos ambientes distribuídos, estudar e averiguar suas possibilidades, e contribuir com o amadurecimento em pesquisas desse campo.

O sistema elétrico de distribuição é um ambiente que tem as características citadas anteriormente. O atual desenvolvimento para uma nova geração destes sistemas (os *smart grids*) aponta para a necessidade de algoritmos que possibilitem a obtenção de dados e o cálculo distribuído (Brown, 2008), criando assim um sistema mais robusto, tolerante a falhas, possibilidade de concorrência no processamento, entre outras características dos sistemas distribuídos (Coulouris et al., 2005).

Propomos neste artigo uma adaptação do algoritmo de Dijkstra, distribuindo-o para execução no modelo de grafo, através do uso de sistemas multiagentes. Espera-se nesse primeiro momento que a proposta consiga solucionar os problemas para topologias estáticas com a mesma qualidade que o algoritmo de Dijkstra convencional consegue atingir.

Este trabalho se constitui das seguintes partes: a Seção 2 descreve o algoritmo de Dijkstra, suas características e seus principais passos. A Seção 3 descreve o que são sistemas multiagentes e sua característica distribuída. A Seção 4 apresenta a proposta de algoritmo distribuído de Dijkstra. Por fim, a Seção 5 comenta sobre a implementação computacional e os testes executados e, em seguida, a Seção 6 conclui o artigo com algumas observações e perspectivas para trabalhos futuros no tema.

2 Algoritmo de Dijkstra

O algoritmo de Dijkstra é um clássico algoritmo computacional para o cálculo da distância mínima entre um vértice e todos os demais vértices de um

grafo. Foi desenvolvido pelo premiado[1] matemático holandês Edsger Dijkstra (1930 - 2002) e publicado na edição número 1 do periódico *Numerical Mathematics*, em 1959 (Dijkstra, 1959).

O Algoritmo 1 apresenta na forma de pseudocódigo o algoritmo de Dijkstra.

Algoritmo 1: Algoritmo de Dijkstra

Entrada: G, v_i

1: Q = Inicializar-Custos-Relativos(G)
2: A = Inicializar-no-Anterior()
3: $S = \emptyset$
4: $R = \emptyset$
5: $D = \emptyset$
6: $temp = 0$
7: $noAvaliado = v_i$
8: **enquanto** $S \neq V$ **faça**
9: $\quad Q$ = Calc-Custo-Relativo($G, noAvaliado$)
10: $\quad S = S \bigcup noAvaliado$
11: $\quad temp = noAvaliado$
12: $\quad noAvaliado$ = Minimo($(V - S), Q$)
13: $\quad A_{noAvaliado} = temp$
14: **fim enquanto**
15: **para todo** $v \in V$ **faça**
16: $\quad D = D \bigcup (v, A_v)$
17: **fim para**
18: $R = (V, D)$
19: **retorne** R

Define-se um grafo $G = \{V, E\}$, onde V é o conjunto dos vértices e E o conjunto das arestas; w será a função custo de cada aresta, onde $\forall w(u, v) >= 0$, tal que a aresta $(u, v) \in E$, $\forall u, v \in V$. Tem-se também um vértice v_i que será o vértice-fonte ao qual se quer calcular a distância mínima entre ele e todos os demais vértices do grafo (os vértices-alvo).

Durante a execução do algoritmo de Dijkstra, cada vértice terá uma variável chamada custo relativo (que simbolizaremos como Q) que será a distância total, naquela iteração, entre este vértice e o vértice-fonte. Estes custos começarão com um valor tão grande quanto possível[2], exceto o custo relativo do vértice-fonte que será 0. Também haverá para cada vértice uma variável chamada vértice anterior (que chamaremos de A), a qual guardará a referência para o vértice em que está ligada naquela iteração.

O algoritmo manterá também um conjunto de vértices S que indicará os vértices que já tiveram suas vizinhanças analisadas pelo método. Durante a execução, o algoritmo selecionará o vértice $u \in V - S$ com o menor custo relativo naquela iteração para ter a sua vizinhança analisada, até que o critério $S = V$ seja atingido, indicando que todos os vértices foram avaliados e o valor da distância mínima entre o vértice-fonte e cada vértice-alvo do grafo foi obtido (Cormen et al., 2009).

De forma descritiva, podemos visualizar o funcionamento do algoritmo de Dijkstra a partir da seguinte sequencia de operações. Passando como argumento o grafo G e o vértice-fonte v_i pertencente ao grafo, inicializam-se variáveis chamadas custos relativos (Q) para cada vértice do grafo com o valor ∞, excetuando-se o custo relativo do vértice v_i que será igual a 0. No algoritmo, este passo corresponde à chamada de função Inicializar-Custos-Relativos, na linha 1. Inicializam-se também as variáveis A durante a chamada da função Inicializar-no-Anterior, linha 2, sendo $A_{v_i} = 0$ e para os demais casos um valor que represente a falta de um antecessor. Na Figura 1 (a), o vértice $v1$ terá sua vizinhança avaliada. Em todas as figuras aqui apresentadas, os vértices que estiverem circulados representarão aqueles que serão avaliados nesta iteração. As arestas pontilhadas representam as ligações entre os vértices, já as contínuas significam aquelas que formam a solução (ou fragmento de solução) para aquela iteração.

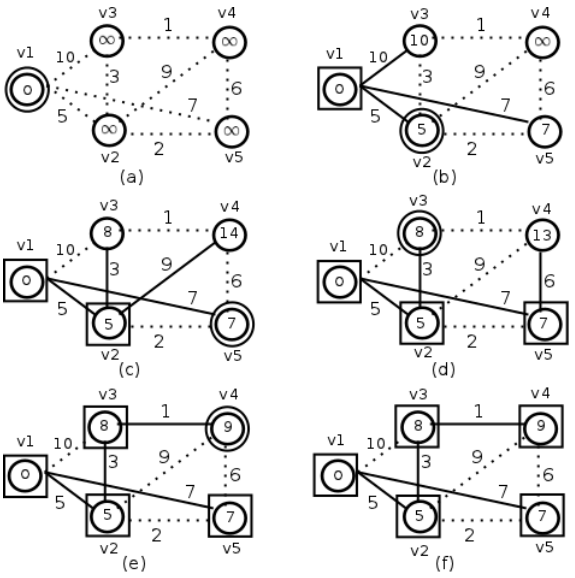

Figura 1: Iterações do algoritmo de Dijkstra. O vértice marcado com um círculo será aquele que terá a vizinhança avaliada naquela iteração; os marcados com um retângulo são aqueles que já tiveram sua vizinhança avaliada.

O algoritmo então fará uma busca na vizinhança de v_i, atualizando o custo relativo de todos os vértices diretamente conectados ao vértice v_i com o respectivo valor da aresta que cria a ligação entre estes dois vértices somado ao valor do custo relativo do vértice v_i. Este passo é simbolizado pela chamada de função Calc-Custo-Relativo. Na Figura 1 (b), a verificação da vizinhança de $v1$ atualizou os custos relativos de $v2, v3$ e $v5$.

[1] Dijkstra foi laureado em 1972 com o Prêmio Turing, considerado o "prêmio Nobel da computação"

[2] Utilizaremos neste texto um "valor infinito" para este propósito.

Agora, marca-se o vértice v_i como um vértice que já teve sua vizinhança avaliada e executa-se outra iteração de avaliação de vizinhança, desta vez no vértice com menor custo relativo atual. Marcaremos os vértices que já tiveram suas vizinhanças avaliadas com o uso de um retângulo.

Caso durante a iteração algum vértice vizinho ao vértice avaliado tenha seu custo relativo atual maior que a soma da aresta que o conecta mais o custo relativo do vértice avaliado, o custo relativo deste vértice vizinho será então atualizado para o valor dessa soma. Isso acontece, por exemplo, com o nó $v3$ onde, na Figura 1 (b) o valor de seu custo relativo é 10 e na Figura 1 (c) é 8. Também ocorre o mesmo com o vértice $v4$ quando o valor do seu custo relativo muda de 14 para 13 nas Figuras 1 (c) e 1 (d) e finalmente assume o valor 9 a partir da Figura 1 (e).

O algoritmo de Dijkstra executa estas iterações até que todos os vértices $v \in V$ do grafo tenham sido selecionados para terem suas vizinhanças avaliadas. Neste momento, temos o caminho mínimo entre o vértice-fonte e todos os demais vértices do grafo. No exemplo, a Figura 1 (e) é o momento em que o nó $v5$ é selecionado para avaliação. No momento seguinte, a Figura 1 (f), representa a solução para o problema nesta instância já com a vizinhança do vértice $v5$ avaliada.

A instrução que começa na linha 15 do Algoritmo 1 indica que, neste momento, serão colocados no conjunto D as arestas que ligam um vértice v à seu antecessor A_v, $\forall v \in V$. Em seguida, na linha 18 é construído o grafo R que será a resposta para o problema, a partir do conjunto de vértices V e de arestas D.

O algoritmo de Dijkstra utiliza uma abordagem gulosa (Coulouris et al., 2005) onde ele sempre avalia a vizinhança do vértice com o menor custo relativo para aquela iteração. Entretanto, ao contrário de outros algoritmos que utilizam técnicas gulosas, o algoritmo de Dijkstra sempre encontra a solução ótima para o problema (Cormen et al., 2009).

O algoritmo de Dijkstra tem ampla aplicação a problemas encontrados no cotidiano de empresas e organizações em geral. Problemas de complexidade polinomial (Cormen et al., 2009) que podem ser modelados a partir de grafos e para os quais a solução seja dada pelo percurso mínimo entre os vértices, permite a utilização do algoritmo de Dijkstra para determinar a resposta. Podemos citar, como exemplos, o algoritmo *Open shortest path first* - OPSPF, que faz o roteamento de pacotes em redes de internet utilizando métodos baseados no algoritmo de Dijkstra (Fortz and Thorup, 2000); a modelagem da cadeia de produção de uma empresa (Arkin and Silverberg, 1987); encontrar o caminho mínimo de reparação de um sistema elétrico de distribuição (Hu et al., 2009); otimizar um sistema de "caronas" em tempo real (Sghaier et al., 2010), entre outros.

A disciplina de algoritmos ensina que, à medida que aumenta-se em magnitude o tamanho da instância do problema que se está trabalhando, aumenta-se também o custo computacional para resolvê-lo (Coulouris et al., 2005).

Apesar do custo computacional ainda ser um problema a ser ponderado no projeto de sistemas e algoritmos, um conjunto de características dos atuais sistemas computacionais também está ganhando importância na avaliação durante a fase de projeto destes sistemas. Os atuais ambientes de computação e informação são distribuídos, de grande magnitude, abertos e heterogêneos (Weiss, 2000). Nestes ambientes, os computadores não são mais sistemas isolados que computam as informações e imprimem respostas. Eles estão cada vez mais interligados com outros computadores ou equipamentos, e tanto o processamento de tarefas quanto a própria obtenção de dados acontece de maneira distribuída (Coulouris et al., 2005).

Neste contexto, sistemas distribuídos acrescentam interessantes possibilidades, como a própria distribuição do processamento ou o melhor acompanhamento do estado de determinado sistema. Entretanto, há muitos desafios também. Por exemplo, modelar como será a iteração entre os componentes do sistema, que protocolo de comunicação será usado, como garantir se o sistema é robusto e tolerante à falhas, entre outras.

A partir deste cenário, utilizaremos o conceito de sistema multiagente para distribuir a execução do algoritmo de Dijkstra e utilizar as vantagens e possibilidades dos sistemas distribuídos.

3 Sistemas Multiagentes

Sistemas multiagentes são um ramo de estudos relacionados com a inteligência artificial distribuída que, diferentemente da inteligência artificial "clássica", alicerça suas pesquisas na possibilidade de aprendizado e resolução de problemas a partir de um fenômeno social (Wooldridge, 2009). Ou seja, o "aprendizado" em um sistema multiagente se dá com maior ênfase nas relações sociais entre os agentes que compõem o sistema.

Agentes são entidades computacionais autônomas, que percebem o ambiente em que estão inseridos e agem sobre este mesmo ambiente tentando atingir seus objetivos (Wooldridge, 2009). Sistemas multiagentes podem ser modelados de forma que um dado grupo de agentes partilhe um objetivo comum. Isso fará com que os agentes cooperem a fim de atingir este objetivo.

Entre outras características, agentes de um sistemas multiagentes são restritos em suas capacidades individuais, tanto em nível de percepção quanto atuação sobre o ambiente, por conta de sua característica distribuída. O controle em sistemas

multiagentes é distribuído, os dados são descentralizados e a computação das informações é assíncrona (Weiss, 2000).

Os principais desafios do desenvolvimento de sistemas multiagentes reside na dificuldade em se modelar de que maneira a dinâmica da sociedade de agentes resultará no cumprimento do objetivo proposto. Para isso, é necessário a determinação de quais tipos de ações e quais agentes irão fazê-las, como será realizada a comunicação entre os agentes, quais protocolos serão utilizados, entre outras. Também é necessário projetar um algoritmo para o processamento distribuído, onde agentes poderão processar tarefas de forma paralela e depois, reuni-las numa solução única para o problema. Outro desafio é quanto à obtenção e comunicação de dados de forma distribuída. Pode ocorrer de apenas alguns agentes conseguirem adquirir certo tipo de informação do sistema. Pensar a maneira de partilhar esta informação para os agentes que irão manipulá-las é uma tarefa a ser realizada na fase de projeto do sistema.

Devido à flexibilidade da tecnologia de sistemas multiagentes e a possibilidade de uso combinado dos conceitos de sistemas distribuídos e inteligência artificial distribuída, aplicações que utilizam esta abordagem vem ganhando espaço nos últimos anos tanto na academia quanto na indústria. Citam-se como exemplos o sistema OASIS que controla o tráfego aéreo no aeroporto de Sydney, além de outras referências na área de mercados, sistemas tutores de ensino e mais (Wooldridge, 2009). Outros exemplos de aplicações de sistemas multiagentes em áreas diversas podem ser encontrados em (Weiss, 2000). Em (McArthur et al., 2007a) e (McArthur et al., 2007b) encontra-se uma revisão sobre o uso desta tecnologia em sistemas elétricos de potência.

4 Algoritmo de Dijkstra Distribuído Através de Sistemas Multiagentes

A implementação distribuída do algoritmo de Dijkstra é realizada a partir do conceito e tecnologia de sistemas multiagentes. A partir da topologia do grafo, aloca-se um agente em cada vértice do mesmo. Como o algoritmo de Dijkstra necessita de um vértice-fonte, utilizou-se um modelo organizacional hierárquico dos agentes (Horling and Lesser, 2004) onde podemos classificá-los em dois tipos: o agente ativo que demanda processamento para os demais agentes é o que está alocado no vértice-fonte; os demais agentes são reativos, recebendo demandas de processamento do agente no vértice-fonte e respondendo aquilo que é pedido.

Cada agente tem uma representação completa do grafo em sua memória, que pode ser atualizada pelo envio de mensagens do agente no vértice-fonte caso aconteça a inserção de um novo vértice ou novas arestas no grafo. Esta funcionalidade provê ao sistema a possibilidade de tratar cenários dinâmicos.

Assim que se iniciam os agentes, o agente no vértice-fonte envia uma mensagem para todos os demais agentes requerendo que estes calculem o menor caminho entre o vértice-fonte e o vértice onde se encontra o agente que recebeu a mensagem. Este passo corresponde à chamada da função Enviar-Grafo-Para-Agentes, no Algoritmo 2, onde M representa o conjunto de todos os agentes alocados nos vértices-alvo do grafo.

Algoritmo 2: Ações do Agente Ativo

Entrada: G, M
1: Enviar-Grafo-Para-Agentes(G, M)
2: $N = \emptyset$
3: $R = \emptyset$
4: **enquanto** $N \neq M$ **faça**
5: $R = R \bigcup$ Resposta-Mensagem-Grafo()
6: $N = N \bigcup$ Resposta-Mensagem-Emissor()
7: **fim enquanto**
8: **retorne** R

Em seguida, o agente receptor da mensagem executa o algoritmo de Dijkstra, iniciando a análise de vizinhança a partir do vértice-fonte. Os passos desta implementação estão explicitados no Algoritmo 3.

Algoritmo 3: Algoritmo de Dijkstra no Agente Reativo

Entrada: G, v_i, v_j
1: $Q =$ Inicializar-Custos-Relativos(G)
2: $A =$ Inicializar-no-Anterior()
3: $S = \emptyset$
4: $R = \emptyset$
5: $D = \emptyset$
6: $W = \bigcup v_i, v_j$
7: $temp = 0$
8: $noAvaliado = v_i$
9: **enquanto** $noAvaliado \neq v_j$ **faça**
10: $Q =$ Calc-Custo-Relativo($G, noAvaliado$)
11: $S = S \bigcup noAvaliado$
12: $temp = noAvaliado$
13: $noAvaliado =$ Minimo($(V - S), Q$)
14: $A_{noAvaliado} = temp$
15: **fim enquanto**
16: $temp = v_j$
17: **enquanto** $temp \neq v_i$ **faça**
18: $W = W \bigcup A_{temp}$
19: $D = D \bigcup (temp, A_{temp})$
20: $temp = A_{temp}$
21: **fim enquanto**
22: $R = (W, D)$
23: **retorne** R

O critério de parada do método, neste caso, diferentemente do Algoritmo 1, ocorre quando o vértice a ser avaliado é o vértice onde se encontra

o agente. Neste momento, pode ocorrer da árvore com o caminho entre os dois vértices conter vértices desnecessários, que serviriam para ligar outros vértices ao vértice-fonte.

Para tratar isto, cada vértice do grafo terá uma variável própria que referenciará o vértice com o qual se está ligada naquela iteração, o A. Assim, ao atingir o critério de parada, basta-se percorrer os valores de A nos vértices a partir do vértice-alvo. O caminho será então obtido, sem vértices desnecessários. Este passo corresponde à instrução que começa a partir da linha 17 do Algoritmo 3. Nela, o conjunto W que representa os vértices do menor caminho entre os nós v_i (vértice-fonte) e v_j (vértice-alvo onde se encontra o agente) começa a adicionar os vértices anteriores A_v a partir do vértice v_j. Também, o conjunto D adiciona as arestas que conectam (v, A_v).

Ao atingir o critério de parada da linha 17 do Algoritmo 3, o grafo R formado por (W, D) na linha 22 terá o caminho mínimo entre o vértice-fonte v_i e o vértice-alvo v_j. A Figura 2 (a) apresenta o resultado da aplicação do algoritmo de Dijkstra onde $v_i = v1$ e $v_j = v3$. Na Figura 2 (b), temos o caminho mínimo entre $v1$ e $v3$ após ser retirado os vértices desnecessários para a resposta, no caso, $v4$ e $v5$.

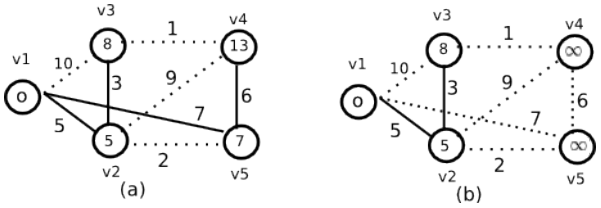

Figura 2: (a) Resultado da aplicação do algoritmo de Dijkstra para $v_i = v1$ e $v_j = v3$. (b) Caminho mínimo entre $v1$ e $v3$.

Com a resposta R obtida, o agente no vértice-alvo a envia para o agente no vértice-fonte, que está esperando as respostas chegarem. Este agente, então, vai sobrepondo cada resposta enviada por cada agente, até que a resposta completa para o grafo como um todo seja obtida. Esta operação se dá na linha 5 do Algoritmo 2, onde o resultado da função Resposta-Mensagem-Grafo é o caminho mínimo entre v_i e algum v_j. Na linha seguinte, o agente marca os demais agentes que já responderam à requisição.

Após todos enviarem suas contribuições, o agente no vértice-fonte termina a execução do algoritmo e o R resultante representa a solução do problema completo.

5 Implementação e Testes Computacionais

Existem vários *frameworks* e linguagens de programação específicas para o desenvolvimento de sistemas multiagentes, cobrindo inclusive diversos paradigmas de programação como a programação lógica, orientada a objetos, estruturada e funcional (Bordini et al., 2006).

Neste trabalho optamos pelo uso do *framework* JADE (*Java Agent DEvelopment*), um conjunto de bibliotecas para a linguagem Java que encapsula diversas funcionalidades e API's para acesso a redes, troca de mensagens, simulação comportamental, entre outras (Bellifemine et al., 2007). Ela possibilita o desenvolvimento completo de um ambiente multiagente a partir da linguagem Java, e tem ampla aceitação na comunidade de pesquisadores em sistemas elétricos de potência (McArthur et al., 2007b).

Para a simulação computacional fora utilizado um conjunto de grafos que variam de 6 vértices e 10 arestas até 48 vértices e 77 arestas. Estes exemplos foram retirados do *website* do professor Ph.D. Kenji Ikeda da Universidade de Tokushima, Japão[3]. Na Tabela 1, a coluna **V** indica o número de vértices da instância; **E** o número de arestas, **S. Ótima** a solução ótima da instância e **S. Encontrada** a solução encontrada pelo algoritmo distribuído de Dijkstra.

Grafos de Teste				
Instância	**V**	**E**	**S. Ótima**	**S. Encontrada**
1	6	10	43	43
2	6	12	25	25
3	8	12	20	20
4	9	15	701	701
5	10	19	253	253
6	12	23	34	34
7	14	22	703	703
8	16	33	23	23
9	16	33	29	29
10	48	77	858	858

Tabela 1: Instâncias de teste.

Percebe-se que, para todas as instâncias, o resultado encontrado pelo algoritmo distribuído de Dijkstra atingiu o valor da solução ótima.

Apesar dos resultados positivos, é importante ressaltar que a principal contribuição desta abordagem reside no fato da adição da característica de processamento distribuído ao sistema. Isso permitiria que o sistema modelado pudesse lidar com topologias dinâmicas, distribuísse o processamento e possibilitasse um controle distribuído sobre o mesmo.

6 Conclusões e Trabalhos Futuros

Os ambientes computacionais modernos são distribuídos, onde diferentes máquinas conseguem tro-

[3]Disponível em `http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/Dijkstra.shtml`, acessado dia 4 de março de 2011

car informações e compartilhar processamento e dados. Neste tipo de ambiente, algoritmos adaptados com estas características são necessários para utilizar as vantagens deste modelo.

Nessa concepção, sistemas multiagentes enquanto ferramentas da inteligência artificial distribuída, tem grande importância para a implementação de sociedades que se comportem de forma distribuída e que aprendam socialmente o comportamento desejado para um dado problema.

Este aprendizado reflete-se na otimização de um dado objetivo a um problema para o qual o sistema multiagente foi modelado, possibilidade de controle distribuído, aferição do estado do sistema entre outras funcionalidades.

Neste trabalhado implementou-se uma versão distribuída do algoritmo de Dijkstra, onde simulamos a alocação de processadores em todos os nós do grafo (utilizando agentes), distribuímos o processamento e combinamos os resultados individuais encontrados para gerar o resultado global.

Com o objetivo atingido de avaliar positivamente o comportamento do algoritmo distribuído comparado à versão convencional em topologias estáticas, imagina-se como trabalho futuro a utilização desta proposta em ambientes dinâmicos, onde a topologia do grafo muda conforme o tempo.

Agradecimentos

Este trabalho foi subsidiado pela CAPES, através de bolsa de pesquisa do programa PROEX. Também tivemos o apoio da FAPESP, através do projeto que financiou o cluster computacional onde executamos os testes.

Referências

Arkin, E. and Silverberg, E. (1987). Scheduling jobs with fixed start and end times, *Discrete Applied Mathematics* **18**(1): 1–8.

Bellifemine, F. L., Caire, G. and Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE*, Wiley.

Bordini, R., Braubach, L., Dastani, M., Seghrouchni, A., Gomez-Sanz, J., Leite, J., O. Hare, G., Pokahr, A. and Ricci, A. (2006). A survey of programming languages and platforms for multi-agent systems, *Special Issue: Hot Topics in European Agent Research II Guest Editors: Andrea Omicini* **30**: 33–44.

Brown, R. (2008). Impact of Smart Grid on distribution system design, *Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, IEEE, pp. 1–4.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2009). *Introduction to Algorithms, Second Edition*, The MIT Press.

Coulouris, G., Dollimore, J. and Kindberg, T. (2005). *Distributed systems: concepts and design*, Addison-Wesley Longman.

Dijkstra, E. W. (1959). A Note on Two Problems in Connection with Graphs, *Numerical Mathematics* **1**: 269–271.

Fortz, B. and Thorup, M. (2000). Internet traffic engineering by optimizing OSPF weights, *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, Vol. 2, pp. 519–528.

Horling, B. and Lesser, V. (2004). A survey of multi-agent organizational paradigms, *The Knowledge Engineering Review* **19**(04): 281–316.

Hu, Y., Chang, Z., Sun, L. and Wang, Y. (2009). Analysis of the Shortest Repaired Path of Distribution Network Based on Dijkstra Algorithm, *2009 International Conference on Energy and Environment Technology*, pp. 73–76.

McArthur, S., Davidson, E., Catterson, V., Dimeas, A., Hatziargyriou, N., Ponci, F. and Funabashi, T. (2007a). Multi-agent systems for power engineering applications - Part I: concepts, approaches, and technical challenges, *Power Systems, IEEE Transactions on* **22**(4): 1743–1752.

McArthur, S., Davidson, E., Catterson, V., Dimeas, A., Hatziargyriou, N., Ponci, F. and Funabashi, T. (2007b). Multi-agent systems for power engineering applications - Part II: technologies, standards, and tools for building multi-agent systems, *Power Systems, IEEE Transactions on* **22**(4): 1753–1759.

Sghaier, M., Zgaya, H., Hammadi, S. and Tahon, C. (2010). A distributed dijkstra's algorithm for the implementation of a Real Time Carpooling Service with an optimized aspect on siblings, *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pp. 795–800.

Weiss, G. (2000). Multiagent Systems and Distributed Artificial Intelligence, *Multiagent systems: a modern approach to distributed artificial intelligence*, Weiss, G., The MIT press.

Wooldridge, M. (2009). *An introduction to multiagent systems*, Wiley, England.