Otimização paramétrica através de busca Hill Climbing distribuída em sistemas multiagentes

Heitor A. Vieira, Rafael Salema Marques, Paulo André L. de Castro

Divisão de Ciência da Computação (IEC)
Instituto Tecnológico de Aeronáutica (ITA)

heitor.a.vieira@gmail.com, salemarsm@gmail.com, pauloac@ita.br

Abstract. *Many problems faced by intelligent systems can be modeled as search problems. One such problem is the definition of parameters of algorithms to maximize (or minimize) an objective function. This procedure demands high cost of time to determine the best values for each parameter. We describe in this paper an autonomous multi-agent system that makes use of the Hill Climbing algorithm exploiting the parallelization of natural processing in multi-agent systems. Several experiments with the classical problems of experience and a parametric optimization on a financial market operator agent were performed. The results demonstrate the effectiveness and efficiency of the system.*

Resumo. *Muitos problemas enfrentados por sistemas inteligentes podem ser modelados como problemas de busca. Um desses problemas é a definição de parâmetros de algoritmos de modo a maximizar (ou minimizar) uma função objetivo que demanda alto custo de tempo para determinar os melhores valores para cada parâmetro. Descrevemos, neste trabalho, um sistema multiagente autônomo que faz uso do algoritmo Hill Climbing explorando a paralelização de processamento natural em sistemas multiagentes. Foram realizados diversos experimentos com o problemas clássicos e uma experiência de otimização paramétrica em um agente operador do mercado financeiro. Os resultados obtidos demonstram eficácia e eficiência do sistema.*

1. Introdução

Em diversos problemas computacionais a solução consiste simplesmente em encontrar uma configuração aceitável, ignorando o caminho para se chegar à tal solução. Sendo assim, este tipo de problema pode ser incluído nos casos em que, definido um estado inicial, a resolução busca encontrar uma resposta, mesmo que esta não seja a mais satisfatória. Nesse contexto, a fim de otimizar os resultados, surgem os algoritmos de melhoria interativa.

A abordagem dos algoritmos de melhoria interativa pode apresentar bons resultados em alguns problemas, sendo capaz de progredir rapidamente de uma solução arbitrária para uma solução mais conveniente.

O presente trabalho propõe um sistema que facilita o uso do algoritmo de melhoria interativa Hill Climbing [Russell and Norvig 2003] em sistema que necessitam

determinar parâmetros para uma função objetivo multivariável que se deseja maximizar (ou minimizar). Utilizou-se na validação agentes autônomos do sistema AgEx [Castro and Sichman 2009], que simula o comportamento dos agentes econômicos no mercado de ações. Foram realizados adicionalmente experimentos de otimização paramétrica para problema clássicos como N-Rainhas e Problema do Caixeiro Viajante [Russell and Norvig 2003].

O estudo do comportamento dos agentes econômicos no mercado de ações pode gerar uma grande demanda de computação. O simulador de agentes competitivos AgEx não foge a esta regra. Dessa maneira, cada sessão de simulação gasta muito tempo de máquina. Ao se tentar realizar uma otimização dos parâmetros presentes em cada agente para que obtenham o maior lucro possível, percebe-se que são necessárias várias sessões de simulação. Então, fez-se necessário um projeto de otimização para este problema.

O presente trabalho propõe um Sistema Multiagentes [Wooldridge 2009] baseado na plataforma JADE [Bellifemine, Caire and Greenwood 2007] para realizar buscas através do algoritmo Hill Climbing [Russell and Norvig 2003]. Este sistema pretende otimizar buscas por um ponto de máximo (ou mínimo) em um domínio de uma função objetivo multivariável.

2. Fudamentos

O algoritmo Hill Climbing é uma técnica de busca local [Russell and Norvig 2003]. O algoritmo começa com uma solução arbitrária do problema e segue iterativamente tentando encontrar um máximo ou mínimo. A cada passo, uma das dimensões do domínio da função estudada é incrementada de um certo Δx. Na implementação do algoritmo, define-se de quanto deve ser o incremento.

Uma variante do *Hill Climbing* é a busca no espaço de soluções em paralelo através do uso de vários agentes autônomos com início definido aleatoriamente. Utilizou-se nesse trabalho tal versão de modo a explorar mais rapidamente o espaço de busca a fim de se obter uma solução rapidamente e continuar a buscar soluções de maior qualidade com o decorrer do tempo. De fato, o *Hill Climbing* é um algoritmo chamado "anytime algorithm". Isso significa que ele pode retornar uma solução válida para o problema em questão mesmo se for interrompido antes de seu término. Ou seja, ele sempre possui um valor ótimo que foi encontrado até então. Espera-se que o algoritmo, antes de encontrar a melhor solução, vá encontrando soluções cada vez melhores.

2.1. Modelo Conceitual

O sistema é composta por agentes de dois tipos: *Hill Climbing* Agent, ou HCA, e o *Hill Climbing Controller*, ou HCC. O HCA é um agente que tem por objetivo encontrar máximos ou mínimos da função objetivo apenas para uma parte do espaço de busca completo. A divisão do espaço de busca é realizada pelo *Hill Climbing Controller*

O HCC possui as seguintes atribuições: instanciar um número pré-definido de HCA, distribuindo-os em várias máquinas através de um esquema Round-Robin [Russell and Norvig 2003] e ouvir mensagens dos HCA que informam o encontro de um ponto de máximo (ou mínimo) local e atribuir ao HCA outro ponto de início. Desta forma, é possível requisitar ao sistema que informe quando encontrar um valor de máximo ou mínimo que esteja dentro de uma determinada faixa especificada

anteriormente pelo usuário. Também é possível que o sistema, caso o usuário não tenha conseguido ainda nenhum indicativo do valor de máximo ou mínimo, permaneça em execução continuamente mostrando o melhor valor encontrado até aquele momento da execução. A Figura 1 apresenta o relacionamento entre o *Hill Climbing Controller* e os *Hill Climbing Agents*.

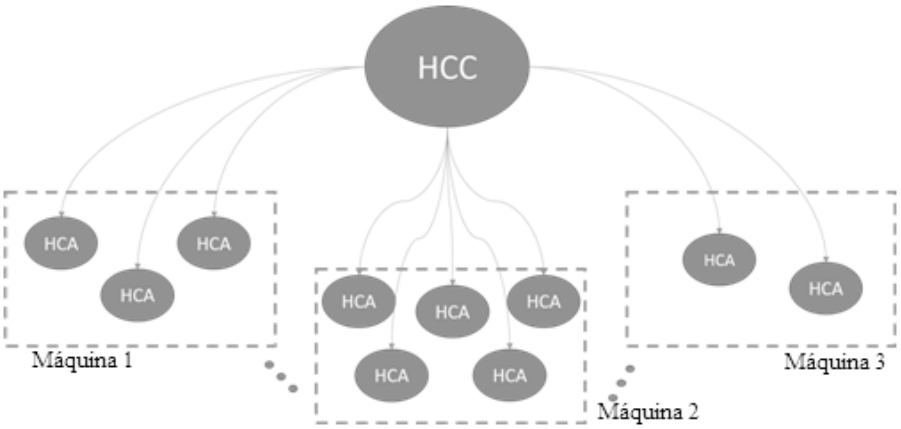

Figura 1. Relacionamento entre HCAs e HCC

2.2. Implementação

O sistema proposto foi implementado utilizando linguagem Java e a plataforma JADE [Bellifemine, Caire and Greenwood 2007] que é compatível com o padrão de comunicação preconizado pela FIPA [6]. apresenta dois tipos de agentes: HCA e HCC. Estes agentes são descritos com mais detalhes a seguir.

O HCA (*Hill Climbing Agent*): é implementado como uma classe Java que estende a classe *Agent* do JADE. Tem a responsabilidade de receber uma mensagem do *Hill Climbing Controller* para efetuar o cálculo da função objetivo considerada. Executa o algoritmo de busca local selecionado, que por default é o *Hill Climbing*, bem como executa o código auxiliar que define a função objetivo. Quando do término deste procedimento, o HCA deve enviar para o HCC uma mensagem de retorno contendo o valor e o ponto do domínio no qual o valor de máximo ou mínimo foi encontrado. Para que o HCA fosse capaz de realizar sua tarefa, foi implementado um Comportamento (*behaviour*) chamado FindLocalMaximumBehaviour que estende a classe OneShotBehaviour do JADE. O método action() desta classe é o responsável por chamar o método da estratégia de busca que efetivamente realizará a procura pelo ponto ótimo. Logo antes do término deste comportamento, um novo *Behaviour*, chamado HCAMessageHandler, é adicionado à fila de comportamentos do agente HCA. Neste momento o HCA aguarda uma mensagem de reset do HCC para começar uma nova sessão de busca.

O HCC (*Hill Climbing Controller*): é implementado como uma classe Java que estende a classe Agent do JADE. Esta classe possui várias responsabilidades. Ela é responsável por requisitar para a plataforma JADE quantos containers conectados ao Main Container existirem na plataforma, bem como uma lista com os identificadores destes. Com esta informação é possível distribuir os HCAs de maneira equilibrada entre os vários Containers distribuídos na rede. A distribuição dos agentes HCAs é feita por

meio do algoritmo Round Robin que está implementado no método roundRobinContainer() no HCC. O HCC possui o método newSession() que é utilizado quando se deseja realizar várias execuções da busca paramétrica com o intuito de tomar várias medidas de tempo de execução e estabelecer uma média do tempo de computação que está sendo gasto.

O HCC também é responsável pela divisão do espaço de busca do problema global entre os vários HCAs, que verão apenas uma pequena parte deste espaço. Deste modo, cada HCA verá apenas um espaço de busca reduzido, levando, portanto, menos tempo de computação para terminar sua busca. Esta divisão é feita pelo método domainDivision(). Além disso, a cada término de sessão é necessário matar os agentes presentes na plataforma. Isto é feito pelo método killAgent() que recebe como parâmetro o AID (Agent Identifier) do agente a ser morto.Após o HCC receber uma mensagem de algum HCA contendo a informação de máximo ou mínimo encontrado aquele é responsável por enviar uma mensagem de reset com um novo valor de ponto inicial para o algoritmo Hill Climbing. O método que faz isso é o sendResetMessage(), que precisa apenas do AID do agente destinatário.

O HCC possui apenas um comportamento: o HCCMessageHandler que estende a classe *Behaviour* do JADE. Este *Behaviour* é responsável por receber as mensagens que os HCAs enviam e verificar se foi achado um valor de máximo ou mínimo dentro do intervalo de aceitação do usuário. Enquanto não for encontrado um ponto que satisfaça a condição de máximo ou mínimo, uma mensagem de reset é enviada para o HCA que havia mandado a mensagem com o ponto encontrado. O HCCMessageHandler possui também um método que mata todos os agentes HCAs da plataforma, para isso chama em um loop o método killAgent() do HCC para todos os HCAs distribuídos nos vários containers presentes na plataforma.

Muitos outros componentes fazem parte do *framework*. A seguir são descritos resumidamente quais são estes componentes e o que fazem.

A classe ParameterSet implementa a classe Concept do JADE. É necessário que haja esta implementação, pois desta maneira é possível passar em uma mensagem um objeto pela plataforma JADE. Esta classe é a responsável por representar o espaço de busca ou domínio da função que esteja em processo de otimização. Um objeto desta classe possui a quantidade de parâmetros da função, bem como os valores máximos e mínimos que cada parâmetro pode assumir.

O pacote hc.constraint abriga classes que definem o domínio de cada problema de busca a ser resolvido. Desta forma, tem-se o desacoplamento da implementação do problema e do sistema de busca distribuído. Cada tipo de problema apresentará uma maneira diferente de locomoção do HCA no domínio. Com o intuito de possibilitar esta diferenciação, foi criada a interface Constraint. Esta interface possui os métodos changeParameter() e createNewRandomParameterSet().

Cada domínio é descrito para os agentes através de uma classe que implemente a interface Constraint, de modo a ser possível mudar o ponto em análise do domínio da maneira adequada para o problema, bem como criar um novo ponto inicial aleatório para o reinício do problema.

Dado que a criação dos objetos do tipo Constraint dependerá do tipo do problema, percebeu-se que o Design Pattern (padrão de projeto) factory (fábrica) seria muito adequado para este caso. Por isso foi criada a classe ConstraintFactory. Nela há um parâmetro do tipo enum que identifica os possíveis tipos de constraints conhecidas. Foram implementados nesse trabalho cinco constraints, cada um responsável pela implementação de um domínio de problema, a saber:

- INT (Integer): Para a resolução de problemas com domínio inteiro de modo genérico;
- TSP (Travel Salesman Problem): Para a resolução do problema do caixeiro viajante;
- NQP (N Queens Problem): Para a resolução do problema das "n" rainhas;
- RFP (Real Function Problem): Para a resolução de problemas de funções de domínio real. Sendo que a discretização do domínio é feita por meio do uso de um passo na busca; e
- TP (Trader Problem): Para a resolução dos problemas relacionados com a otimização dos comerciantes de ações do AgEx.

Para cumprir o padrão de comunicação FIPA foi criado o pacote ontology. Nele estão definidos tanto a Ontologia quanto o Vocabulário utilizados no projeto. A interface HillClimbingVocabulary é utilizada para definir os termos utilizados pelos agentes na comunicação. Alguns termos são: MAXIMUM, CONSTRAINT, OBJECTIVE_FUNCTION, PARAMETERSET, RESET.

A classe HillClimbingOntology estende a classe Ontology do JADE e implementa a interface HillClimbingVocabulary. A ontologia tem por objetivo acrescentar novos esquemas de estruturas de dados para que a plataforma JADE possa entender os objetos que estão sendo enviados e recebidos pela rede. Dessa maneira é possível, por exemplo, acrescentar a definição de um conjunto de parâmetros, ou parameterSet, por meio da adição de esquemas básicos como o número de parâmetros como um tipo inteiro e os máximos e mínimos de cada parâmetro como um vetor de floats.

O pacote hc.objectiveFunction é também de extrema importância ao projeto. É nele em que está definida a interface ObjectiveFunction que estende a classe Concept. Esta interface possui os seguintes métodos principais de definição de comportamento:

- getObjetiveFunctionValue(ParameterSet specification): Este método tem como objetivo definir como comportamento o retorno do valor da função dado o ponto do domínio especificado por um objeto da classe ParameterSet.
- getNumberOfParameters(): Este método apenas retorna a quantidade de parâmetros do espaço de busca, ou seja, retorna a dimensão do domínio da função de busca.

Os componentes descritos anteriormente podem ser visualizados de maneira estruturada pelo diagrama de classes simplificado mostrado abaixo pela Figura 2.

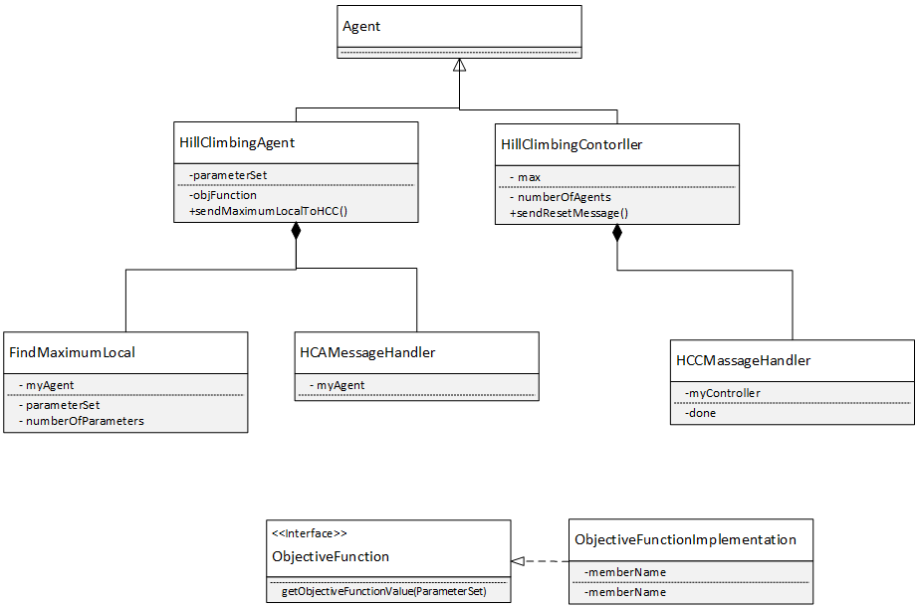

Figura 2. Diagrama com as principais classes do sistema

3. Experimentos e resultados

Conforme descrito anteriormente, o objetivo do trabalho é construir uma plataforma capaz de realizar de forma distribuída por parâmetros ótimos de uma função objetivo. Foram realizados experimentos clássicos descritos em 3.1. O sistema foi utilizado ainda na otimização paramétrica de um agente que opera de modo autônomo no mercado financeiro. Tal agente, RSI, foi implementado em um simulador de mercado financeiro, chamado AgEx. Estes experimentos e seus resultados são apresentados a seguir.

3.1. Problemas clássicos

Foram selecionados dois problemas clássicos da área de Inteligência artificial, N-Rainhas e o Problema do Caixeiro Viajante, descritos por [Russell and Norvig 2003], que foram implementados de forma distribuída para servir como estudo de caso.

3.1.1. Problema do Caxeiro Viajante (TSP)

O problema do Caixeiro Viajante (*Travel Salesman Problem*) é um problema de determinação da menor rota possível que percorra uma série de cidades passando uma por vez, sem repetição, e retornando a cidade de origem. Este problema é um caso de otimização do tipo NP-completo.

No caso estudado modelou-se o problema de modo que havia na base de dados apenas a localização de cada cidade em coordenadas cartesianas. Dessa maneira a estrutura de dados que se tem é uma lista contento pontos de um plano que representam as cidades. A fim de simplificar a presente experiência, considerou-se que sempre existe um caminho em linha reta entre duas cidades quaisquer dentro do conjunto de cidades estudado.

Este teste contribuiu apenas para verificar o funcionamento da aplicação e do caráter distribuído do sistema, apresentando como resultado a validação do modelo proposto.

3.1.2. Otimização paramétrica das N-Rainhas

O problema das N-Rainhas consiste em dispor um número "N" de rainhas em um tabuleiro N x N de modo que nenhuma rainha esteja em posição de ataque com relação a qualquer outra rainha. Dessa maneira, duas rainhas não podem compartilhar a mesma coluna, linha ou diagonal.

A função objetivo construída para o problema das N-Rainhas retorna a contagem de pares de rainhas que estão atacando uma à outra. Dessa maneira quanto mais próximo de zero for esta função mais próximo de uma solução o framework estará. Logo, a condição de parada da busca ocorre quando é encontrada uma configuração de rainhas cujo custo seja igual a zero.

Na implementação utilizou-se um vetor que armazenava apenas uma das coordenadas da posição de uma rainha, a outra coordenada é dada pela posição no vetor. Tendo em vista que na solução cada rainha irá ocupar uma coluna. É suficiente considerar que distribuiu-se uma rainha em cada coluna e que o valor x presente em cada posição i do vetor representa apenas a linha que a rainha ocupa na i-ésima coluna. Neste caso de teste ainda não se utilizou da divisão de domínio como estratégia de otimização paramétrica. As diferenças de tempo decorrem apenas do fato da existência de vários agentes explorando soluções simultaneamente. Por isso, podem existir casos em que mesmo com maior grau de paralelismo obteve-se menor desempenho.

Várias execuções deste problema foram realizadas. Dois parâmetros foram variados: o número de rainhas, ou seja, o tamanho do problema e o número de HCAs (agentes de busca Hill climbing), o que está relacionado diretamente com o nível de paralelismo da solução. Na Figura 4, apresenta-se um gráfico com os desempenhos de obtidos com as várias configurações para cada tamanho de problema. Considera-se o desempenho da configuração com 1 HCA para cada tamanho de problema como 1 e calcula-se o desempenho das demais dividindo o tempo obtido com 1 HCA pelo tempo obtido na outra configuração para um mesmo tamanho de problema, assim números maiores indicam melhor desempenho. Dessa forma, o desempenho é igual a um para cada tamanho de problema e configuração com 1 HCA. Exceto no caso com 25 rainhas onde não foi possível obter resultado com apenas 1 HCA e o tomou-se como tempo base a configuração com dois HCAs.

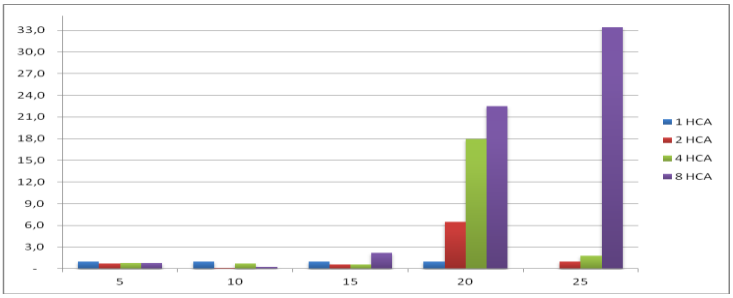

Figura 3. Desempenho do sistema em várias configurações e tamanhos de problema (N-Rainhas)

Pode-se observar um significativo ganho de desempenho com o crescimento do tamanho do problema e do número de HCAs. Isto indica um bom aproveitamento do paralelismo do problema. Entretanto em problemas pequenos, pode-se pequenos ganhos ou mesmo uma queda de desempenho, provavelmente isto ocorre porquê o tempo de computação da solução é extremamente rápido e o tempo de execução total é composto majoritariamente pelo *overhead* de trabalho. Isto quer dizer que os valores de tempo para n de 5 a 15 não são significativos para uma conclusão relativa à otimização paramétrica. Com o incremento do tempo de computação, os casos em que o valor de n é 20 e 25 já são mais interessantes. Percebe-se claramente uma evolução no desempenho à medida que aumenta-se o número de HCAs, em outras palavras, foi observada melhorias com o aumento do paralelismo da solução.

3.2. Otimização paramétrica no AgEx

Para mostrar a otimização no desempenho da busca paramétrica em agentes foi estabelecido que cada simulação do AgEx deveria possuir apenas um agente econômico, pois desta maneira poderiam ser evitados efeitos indesejados causados por agentes interferindo na execução do outro. Considerou-se também que o agente iria se concentrar em comprar e vender ativos de apenas uma empresa.

Para os casos de teste, foram tomados os valores dos papéis da Microsoft (MSFT) no período de primeiro de Janeiro de 2009 até 28 de fevereiro de 2009. Com isso, obteve-se um total de 40 dias úteis, ou seja, 40 passos de clock do AgEx por simulação.

De maneira semelhante aos casos de teste, para conseguir estabelecer uma medição de tempo de execução, utilizaram-se agentes econômicos com funções de custo conhecidas. Foi escolhido o agente RSI (*Relative Strength Index*), ou Índice de Força Relativa (IFR).

O agente RSI é um agente que utiliza uma estratégia baseada em uma medida conhecida como índice de força relativa. Este índice foi publicado por J. Welles Wilder, em 1978, no seu livro intitulado New Concepts in Technical Trading Systems. O RSI é classificado como um oscilador de momento, medindo a velocidade e magnitude dos movimentos direcionais de preços. O momento é definido como a frequência das quedas e subidas de preços, ou seja, mede o quão estável ou oscilantes estão os preços do papel.

No agente RSI foram manipulados dois parâmetros: o período e o *threshold*. O período é um parâmetro que determina em qual janela de tempo o agente irá considerar seu cálculos. Se o valor deste período for 10, por exemplo, significa que irá considerar os valores de fechamento do papel pelos 10 dias predecessores ao presente dia, para poder tomar decisão de compra ou venda do mesmo. O *threshold* é a variável que mensura o índice de força relativa do papel, ou seja, a medida da velocidade e magnitude dos movimentos de preço, conforme explicado anteriormente.

Por meio de várias execuções do AgEx, foi possível mapear parte dos valores da função lucro que o agente RSI obtém para a janela de tempo de simulação utilizada. A Figura 5 a seguir ilustra o lucro obtido com a experiência.

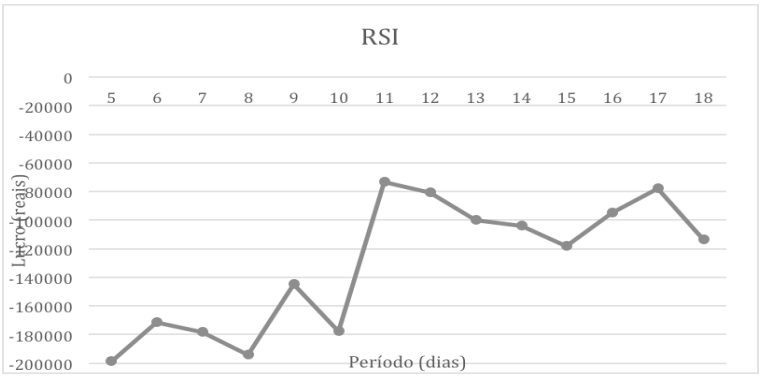

Figura 4. Lucro do agente RSI variando o período com *threshold* valendo 70

Na função lucro do RSI criou-se o período de tempo considerado para cálculo da média e mantendo constante o valor do *threshold* em 70. Percebe-se que a função se comporta de maneira suave, ou seja, sem saltos entre valores consecutivos. Existem dois máximos locais nos períodos 11 e 17.

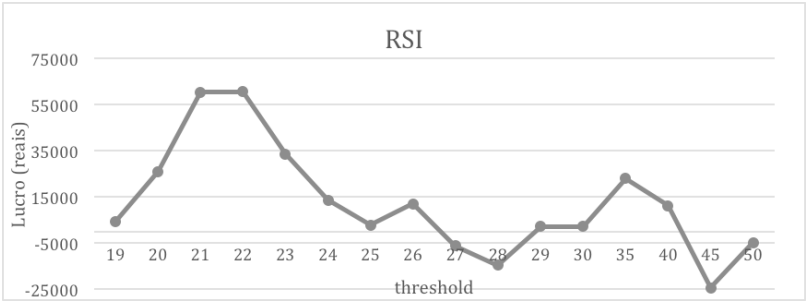

Figura 5. Lucro do agente RSI variando o *threshold* com período de 11 dias

A Figura 6 traz o gráfico do lucro do agente RSI variando-se o *threshold* e utilizando 11 dias como valor do período. Percebe-se que o gráfico apresenta um aspecto. Existe uma forma parecida com um platô nos valores 21 e 22 de *threshold*, entretanto existe uma diferença pequena de valores sendo o lucro maior quando o *threshold* vale 22.

Tabela 1. Valores de tempos para a otimização paramétrico do agente RSI

Número de HCAs	Tempo (ms)	SpeedUp
1	1.333.028,0	1
2	743.538,0	1,79
4	380.828,0	3,50

Sendo assim, percebe-se uma melhoria de desempenho na busca pelo parâmetro ótimo. Para dois HCAs houve uma melhoria de 1,79 e para 4 HCAs houve uma melhoria de 3,5. Valores bem próximos de dois e quatro, que seriam as melhorias ideais.

4. Conclusões

Este trabalho propõe um sistema que otimiza as buscas paramétricas por pontos ótimos nos lucros dos agentes econômicos do AgEx, utilizando o Hill Climbing como

algoritmo de busca local padrão. Este sistema fornece uma plataforma multiagente capaz de efetuar buscas paramétricas para qualquer tipo de problema em que seja possível a modelagem por meio de uma função custo real de variáveis reais com qualquer algoritmo de busca local de forma distribuída de modo a explorar o paralelismo natural em algoritmos de busca local.

O sistema foi testado com três domínios de problemas distintos. Dois deles problemas clássicos usados em computação: N-Rainhas e Problema do Caixeiro Viajante (TSP). O terceiro domínio foi a otimização paramétrica de um agente que opera de modo autônomo no mercado financeiro.

Em termos de implementação, utilizou-se a plataforma de sistemas multiagentes JADE e o padrão de comunicação entre agentes sugeridos pela FIPA. Essa escolha facilitou a comunicação entre os agentes componentes do sistema aqui proposto: HCA e HCC mesmo quando estavam localizados em máquinas distintas. Desta forma, o produto final mostrou-se flexível o suficiente para cobrir várias configurações de cenários de buscas paramétricas.

Para trabalhos futuros, recomenda-se a testar a robustez do sistema quanto a possíveis problemas de rede ao fazer uso de um grande número de HCAs estiver presente. Como existe a possibilidade do HCC tornar-se um gargalo do sistema, sugere-se estudar a criação de uma hierarquia de HCCs. Procedimento que pode contribuir para otimizar o trabalho de comunicação e coordenação dos HCAs.

Referências

Bellifemine, F., Caire, G. and Greenwood, D. (2007) “Developing multi-agent systems with JADE”, 1. ed., John Wiley and Sons Ltd., England.

JADE: Java Agent DEvelopment Framework, (2013). Disponível em: <http://jade.tilab.com/>. Acesso em: Outubro de 2014.

Castro, P. A. L. and Sichman, J. S. (2009) “AgEx: a financial market simulation tool for software agents.” In: Aalst W, Mylopoulos J, Sadeh NM, Shaw MJ, Szyperski C, Filipe J, Cordeiro J (eds) LNBIP, vol 24.Springer, Berlin, pp 704–715.

Russell, S. and Norvig, P. (2003) “Artificial intelligence. A modern approach”, 2nd ed., Prentice Hall, Englewood Cliffs.

Wooldridge, M. (2009) “An Introduction to MultiAgent Systems”, 2nd. ed., John Wiley and Sons Ltd., England.

FIPA: The Foundation for Intelligent Physical Agents, (2009). Disponível em: <http://www.fipa.org>. Acesso em: Outubro de 2014.