Contratação de Agentes em Grupos de Larga Escala

Christian J. Delgado Polar[1], **Luiz Chaimowicz**[1]

[1]Departamento de Ciência da Computação – UFMG
Belo Horizonte – MG – Brazil

chris@dcc.ufmg.br, chaimo@dcc.ufmg.br

Abstract. *With the advances in ubiquitous and massive computing systems, there is an increasing need of technologies that deal with large number of agents. Specially, systems of self-interested agents, each one representing different persons or institutions, are gaining importance. In such systems, one agent normally has to negotiate with and contract other agents to fulfill its objectives. This paper presents a scalable protocol to perform this type of contracting in swarms of self-interest agents. The protocol was tested in simulations where teams of robotic vehicles must transport items from specific targets. Results demonstrate that the proposed protocol is scalable, rational and fair, being suitable for this type of system.*

Resumo. *Com o avanço da computação massiva e ubíqua, existe uma necessidade cada vez maior de sistemas que trabalhem com um grande número de agentes. Em especial, ambientes compostos por agentes que não compartilham o mesmo interesse tem recebido cada vez mais atenção. Nesse tipo de ambiente, normalmente um agente deve contratar outros de forma a realizar suas tarefas. Esse artigo propõe um protocolo escalável para a contratação de agentes com interesses próprios em swarms. O protocolo foi testado em simulações onde times de veículos autônomos são contratados para transportar mercadorias. Os resultados obtidos mostraram que o protocolo proposto é escalável, racional e justo, sendo adequado para esse tipo de ambiente.*

1. Introdução

Sistemas compostos por uma grande quantidade de agentes tem se tornado úteis em diversos ambientes. Genericamente chamados de *swarms*[1] esses grupos são formados por agentes mais simples, com capacidades reduzidas de comunicação e processamento mas que em conjunto são capazes de resolver problemas complexos. Com isso, diversas pesquisas na área de sistemas multi-agentes e robótica tem se concentrando em grupos compostos por centenas ou milhares de indivíduos [Konolige et al. 2004, Scerri et al. 2004, Xu et al. 2005, Jang and Agha 2004].

Especificamente, swarms formados por agentes que representam os interesses de diferentes entidades são cada vez mais freqüentes. Um grupo de sensores móveis onde uma ou mais organizações precisam perceber diferentes tipos de informação (prevenção de desastres, estudos climáticos, etc), dois grupos compostos por muitos robôs, um realizando tarefas de limpeza e o outro de vigilância ambos pertencentes a diferentes

[1]O termo "*swarm*" é utilizado aqui para representar um grande grupo de agentes simples. Esse artigo não explora características de comportamento de swarms tais como estigmergia ou auto-organização.

organizações, um grupo composto por muitos robôs e/ou humanos realizando atividades de exploração (mineral, espacial) onde cada indivíduo representa os interesses de uma determinada organização (sub-time) são exemplos de swarms compostos por uma grande quantidade de agentes com interesses próprios.

Neste tipo de sistema, muitas vezes um agente precisa contratar um outro agente quando não está suficientemente capacitado para realizar uma tarefa. A contratação de agentes (*multi-agent contracting*) é um problema interessante que tem sido muito estudado na última década [Kraus 1996, Sandholm and Lesser 1995a, Collins et al. 2002, Fischer et al. 1996, Babanov et al. 2003, Smith 1980]. A contratação de agentes é um processo naturalmente complexo. Conflitos causados, por exemplo, pelo preço do contrato são muito comuns. O processo se torna ainda mais difícil quando não se possui uma visão global do conjunto de possibilidades de negócio, isto é, do conjunto de agentes que podem ser contratados. Nos swarms, devido ao grande número de agentes, somente é possível obter uma visão local do grupo dado que obter informações de todos os agentes é uma operação extremamente custosa.

Dada a grande quantidade de membros do grupo e o dinamismo do ambiente que caracterizam esse tipo de sistema, algoritmos para swarms devem consumir poucos recursos computacionais e requerer níveis baixos de comunicação, ou seja, devem ser escaláveis. Até agora as propostas em contratação de agentes com interesses próprios (*contracting*) tratam de algoritmos que não estão preparados para swarms.

Portanto, o presente trabalho propõe um protocolo escalável que possibilita a cooperação e negociação por meio da realização de contratos entre times com interesses próprios num swarm de agentes. Para isso, o protocolo adapta técnicas para alocação de tarefas em grupos de grande escala propostas em [Scerri et al. 2005] e técnicas de negociação entre agentes com interesses próprios [Sandholm 1999].

2. Definição do Problema

Considere um sistema composto por um swarm formado por vários times $L = \{L_1, ..., L_p\}$ cada um com um interesse diferente, onde cada time é formado por um conjunto de agentes $E_p = \{e_{1,p}, ..., e_{n,p}\}$ e está encarregado de um conjunto de tarefas $\Theta_p = \{\Theta_{1,p}, ..., \Theta_{m,p}\}$. O problema da contratação em sistemas multi-agente de larga escala consiste em encontrar eficientemente, utilizando níveis baixos de comunicação e processamento, um conjunto de contratos para alocar tarefas em agentes: $Cont(e_{i,l}, \theta_{j,p}) \in \{0, 1\}$ (que é 1 se o agente $e_{i,l}$ do time l é contratado para realizar a tarefa $\theta_{j,p}$ do time p e 0 se não é contratado). Os contratos devem maximizar no tempo o ganho total obtido pelos times $\mathcal{G}$, por meio dos ganhos parciais obtidos por todo agente $e_{i,l}$ dentro do conjunto total E de agentes no sistema pela execução de cada tarefa $\theta_{j,p}$ dentro do conjunto total de tarefas Θ:

$$\mathcal{G} = \sum_{e_{i,l} \in E} \sum_{\theta_{j,p} \in \Theta} (Pag(\theta_{j,p}) - (Cus(e_{i,l}, \theta_{j,p}) * Cont(e_{i,l}, \theta_{j,p}))), \quad (1)$$

onde

$$\forall \theta_{j,p} \in \Theta, \textstyle\sum_{e_{i,l}} Cont(e_{i,l}, \theta_{j,p}) \leq 1.$$

$Pag(\theta_{j,p})$ é o pagamento recebido do exterior do sistema e $Cus(e_{i,l}, \theta_{j,p})$ é o custo incorrido pelo agente $e_{i,l}$ para a execução da tarefa $\theta_{j,p}$.

Como em todo protocolo de contratação, a forma de encontrar o conjunto de contratos deve ser, preferivelmente: pareto–eficiente, simples, individualmente racional e distribuída. Um protocolo possui racionalidade individual se utilizar o protocolo é benéfico para os agentes. Um protocolo é benéfico se garante os interesses dos participantes na negociação. Um protocolo é pareto–ótimo se, definido o contrato, não existe um outro contrato que seja preferido por ambos agentes, ou seja, não é possível encontrar um outro contrato onde os ganhos dos dois agentes aumentam.

3. Protocolo para Contratação de Agentes em Swarms (PCS)

O protocolo para contratação em swarms (PCS) proposto neste trabalho permite a cooperação por meio do estabelecimento de contratos entre agentes ou times de agentes com interesses próprios num swarms de agentes. Além disso, o PCS permite também a alocação de tarefas entre agentes que têm os mesmos objetivos dentro do swarm.

O PCS recebe como entrada um conjunto de times L e um conjunto de tarefas Θ e obtém como saída um conjunto de contratos C. O conjunto de times L forma um swarm. A Figura 1 mostra a interação dos agentes envolvidos numa execução simples do protocolo. Cada time distribui suas tarefas em diferentes agentes pertencentes ao time dentro do swarm, esses agentes são chamados de *administradores*. Cada administrador percorre um pequeno subconjunto dos agentes dentro do swarm numa etapa chamada de *descoberta*. O administrador faz requisições de capacidade aos agentes que ele vai encontrando. Esses agentes são chamados de *candidatos*. Em uma etapa de *negociação* o administrador oferece aluguéis até o candidato aceitar. Uma vez que o candidato aceita a oferta, a tarefa é alocada e o contrato executado.

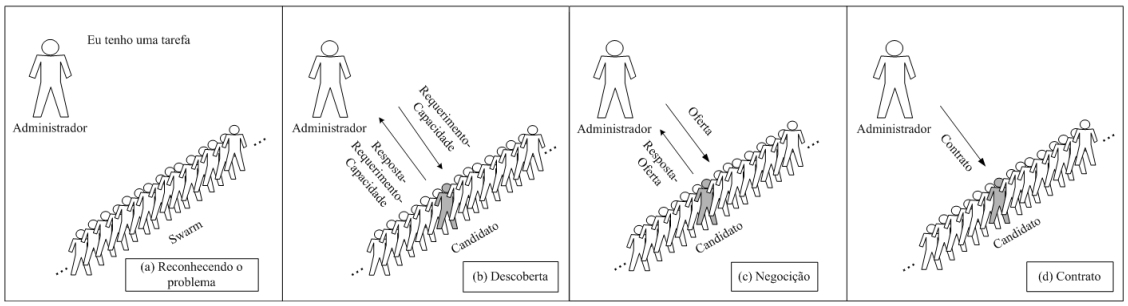

Figura 1. Interação no protocolo para contratação em swarms.

Dentro do PCS, um agente $e_{i,l}$ contrata um outro agente $e_{j,p}$ para a realização da tarefa $\theta_{k,l}$ e paga um aluguel $Al(e_{j,p}, \theta_{k,l})$ (calculado numa negociação) ao agente $e_{j,p}$ uma vez que $\theta_{k,l}$ é executada. $e_{j,p}$ pode pertencer ao mesmo time l ($l = p$) ou a outro time. Quando $e_{i,l}$ contrata um agente $e_{j,l}$ do mesmo time, não é pago aluguel ($Al(e_{j,l}, \theta_{k,l}) = 0$) e também não é realizada nenhuma negociação. Simplesmente, $e_{i,l}$ aloca a tarefa em $e_{j,l}$ se $e_{j,l}$ não possui uma outra tarefa $\theta_{m,r}$ com maior oferta de aluguel $Al(e_{j,l}, \theta_{m,r})$. O PCS tenta encontrar os melhores contratos, ou seja, a melhor alocação de tarefas e agentes. A alocação que o PCS tenta encontrar não somente é eficiente (são executadas as tarefas que possuem maior pagamento e utilizados os agentes melhor capacitados), mas também

justa, ou seja, os times que estão melhor preparados para competir obtêm os melhores agentes. O protocolo permite também a definição de aluguéis justos, isto é, regidos pela oferta e a demanda.

As seções seguintes detalham as etapas do PCS descrevendo os comportamentos dos agentes envolvidos no processo.

3.1. Descoberta

Quando um time p é encarregado de uma tarefa $\theta_{i,p}$, esta é enviada a um agente $e_{k,p}$ qualquer do time. Esse agente, chamado de *administrador* é responsável por encontrar um outro agente candidato $e_{j,l}$ que esteja capacitado para realizar a tarefa.

O processo de descoberta é aproximado. Não se tenta encontrar o melhor agente mas sim um agente que tenha uma capacidade aceitável para realizar a tarefa. Devido à grande quantidade de agentes, não é viável realizar uma busca exaustiva considerando os indivíduos dentro do sistema. Para solucionar esse problema, o PCS utiliza a técnica proposta em [Scerri et al. 2005] onde se faz uso do conhecimento global que se tem do sistema (quantidade de agentes, número de tarefas e distribuição de capacidades) para determinar se um agente é suficientemente capacitado para realizar uma tarefa. O time mantém atualizada a informação global baseado em informação enviada pelos agentes no swarm e passa esse conhecimento aos administradores junto com a tarefa encarregada.

No processo de descoberta, o administrador, primeiramente, cria um pacote de informação (*token*). Um token é uma tupla com quatro elementos: $\Delta_{m,p} =< \theta_{i,p,c}, T_{max}, T_{min}, OfertaMax(\theta_{i,p,c}) >$. O token possui dois threshold T_{max} e T_{min} que determinam a capacidade máxima e mínima que pode ter um agente para poder realizar a tarefa. Os thresholds são calculados no momento da criação do token como será descrito na seção 3.3. $OfertaMax(\theta_{i,p,c})$ é a máxima oferta possível em uma negociação. Depois de criar o token, o administrador escolhe aleatoriamente um agente $e_{j,l}$ dentro do swarm e pede informação que lhe permita saber a capacidade $Cap(e_{j,l}, \theta_{i,p,c})$ do agente para realizar a tarefa $\theta_{i,p,c}$ da classe c. Tarefas com características parecidas são agrupadas em uma mesma classe. Se possui uma capacidade adequada, esse agente é escolhido como candidato para realizar a tarefa. Um agente $e_{j,l}$ é candidato se $T_{min} < Cap(e_{j,l}, \theta_{i,p,c}) < T_{max}$.

O token continua percorrendo agentes (em cada passo o administrador escolhe aleatoriamente um agente vizinho do último agente encontrado) até encontrar um candidato ou até atingir um número máximo de passos. Se um número máximo de passos é atingido, o token sai do sistema. Além disso, o token automaticamente diminui T_{min} depois que atinge um número determinado de passos utilizando esse threshold. Dada a natureza aleatória do processo e dado que a informação global pode ficar desatualizada num ponto da execução do protocolo, o T_{min} inicialmente calculado pode não ser mais adequado para a contratação. A diminuição do T_{min} permitirá novas tentativas para se encontrar um agente capacitado.

Com este processo, a contratação realiza-se de forma distribuída. Durante a contratação, os administradores não precisam se comunicar com nenhuma entidade central ou com o time como um todo para tomar suas decisões. Eles também não precisam se comunicar com todos os agentes no swarm o que centralizaria o processo nos agentes administradores.

3.2. Negociação

Na negociação, os agentes determinam as condições da contratação. No PCS, especificamente, define-se o preço do aluguel do agente candidato. O administrador $e_{k,q}$ tenta encontrar um aluguel $Al(e_{j,l}, \theta_{i,q,c})$ que maximize o ganho líquido $Gl(\theta_{i,q,c}) = Pag(\theta_{i,q,c}) - Al(e_{j,l}, \theta_{i,q,c})$ do time. O aluguel é calculado por leilões de tarefas e agentes. A negociação está baseada em uma mistura de dois tipos de leilões: leilão inglês e leilão holandês [Sandholm 1999].

Como descrito na seção anterior, o administrador, primeiramente, procura um agente $e_{j,l}$ capacitado. Depois calcula um aluguel inicial baseado no menor custo que pode incorrer qualquer agente com $Cap(e_{j,l}, \theta_{i,q,c})$ para realizar a tarefa, o aluguel deve cobrir pelo menos o custo da realização da tarefa ($Al(e_{j,l}, \theta_{i,q,c}) = CusME(e_{j,l}, \theta_{i,q,c})$). O administrador também calcula uma máxima oferta $maxOfertaCandidato$ para $e_{j,l}$ que é uma função do pagamento que será recebido ($maxOfertaCandidato = f(Pag(\theta_{i,q,c}))$). A oferta inicial é enviada ao agente. Se a oferta é aceita $e_{j,l}$ é contratado. Se a oferta não é aceita o administrador começará a procurar outros agentes $e_{h,p}$ com $Cap(e_{h,p}, \theta_{i,q,c}) \approx Cap(e_{j,l}, \theta_{i,q,c})$ com uma tolerância determinada pelo time. Os agentes encontrados serão incluídos dentro do conjunto de $Candidatos$.

Uma vez formado o conjunto de $Candidatos$ o administrador ofertará aluguéis cada vez maiores a todo agente $e_{h,p} \in Candidatos$ até atingir um aluguel igual a $maxOfertaCandidato$. Uma oferta é incrementada com alguma quantidade fixa δ_q definida pelo time ($Al(e_{j,l}, \theta_{i,q,c}) = Al(e_{j,l}, \theta_{i,q,c}) + \delta_q$). Se nenhum candidato aceita nenhuma oferta o administrador formará um outro conjunto de $Candidatos$. Em todo o processo o agente contratado será o primeiro agente a aceitar uma oferta.

Cada vez que o administrador procura um agente, ele primeiro recalcula os thresholds (seção 3.3) para verificar se é possível encontrar ainda um outro agente capacitado e com um custo aceitável. Todo token possui uma oferta máxima que é uma função do pagamento recebido pelo time ($OfertaMax(\theta_{i,q,c}) = f(Pag(\theta_{i,q,c}))$). Um agente $e_{j,l}$ possui um custo aceitável se $Cus(e_{j,l}, \theta_{i,q,c}) <= OfertaMax(\theta_{i,q,c})$. Se não existiram mais agentes com custo aceitável o token sai do sistema.

Esta forma de negociação permite que os administradores possam concorrer primeiro por agentes mais capacitados ao utilizar thresholds iniciais relativamente altos. Eles oferecerão, para agentes bem capacitados, aluguéis inicialmente baixos, até oferecer quase o pagamento total pela realização da tarefa. Também neste processo todos os agentes candidatos receberão uma oferta igual antes do que ela seja incrementada dando a mesma oportunidade a cada candidato de aceitar a oferta.

A fim de poder realizar alocações simples entre agentes do mesmo time utilizando o mesmo algoritmo da negociação entre agentes com interesses próprios, a contratação entre agentes que compartilham o mesmo objetivo é realizada da seguinte maneira: o aluguel ofertado por um administrador $e_{i,l}$ a um candidato $e_{j,l}$ do mesmo time é $Al(\theta_{k,l,c}) = Cus(e_{j,l}, \theta_{k,l,c})$. Quando $e_{j,l}$ avalia suas ofertas o aluguel ofertado pelo administrador será interpretado como $Al(\theta_{k,l,c}) = Pag(\theta_{k,l,c})$. Além disso, esse aluguel não é negociado, ou seja, ele não incrementa e somente é ofertado uma vez. O agente $e_{j,l}$ simplesmente aceita a tarefa se não possui uma outra tarefa com maior aluguel. Administradores do mesmo time também não concorrem em leilões, mas podem sim ofertar a

agentes que já possuem uma oferta do mesmo time em casos onde o pagamento pela sua tarefa seja maior.

Toda oferta permanece aberta para o agente por t_l passos, se o agente não responde durante esse tempo, a oferta é considerada não aceita. Um passo é o tempo que demora qualquer token para ser enviado desde um agente a outro. O tempo t_l é determinado por cada time l em particular. Cada vez que um agente recebe uma oferta maior do que as recebidas até o momento, ele comunica o valor dessa melhor oferta aos administradores que lhe ofertaram alguma vez e que ainda não alocaram suas tarefas. Essa oferta é retida momentaneamente pelo agente e pode ser respondida (aceita ou recusada) em um tempo t entre o tempo atual e o tempo no qual a oferta atingirá a espera máxima ($t_{atual} \leq t \leq t_{atual} + t_l$). Essas duas características do protocolo permitem a formação de um leilão inglês onde os administradores tendo conhecimento da melhor oferta atual podem oferecer um aluguel maior.

Os candidatos, baseados na informação global que possuem, ao avaliar uma oferta, verificam se existem outros agentes que possam aceitar a oferta, ou seja, agentes concorrentes. Se existe pouca concorrência o agente pode recusar a oferta com uma probabilidade inversamente proporcional à quantidade de agentes capacitados que podem aceitar a oferta. Desta maneira os administradores começarão ofertando aluguéis pequenos para ganhar agentes bem capacitados e os candidatos esperarão até níveis altos de concorrência para aceitar uma oferta, isto é os candidatos estarão num leilão holandês concorrendo por tarefas.

3.3. Cálculo de Thresholds

Todo token $\Delta_{i,A}$ possui dois thresholds T_{max} e T_{min} . O cálculo de cada threshold está baseado em dois elementos: a distribuição de capacidades dos agentes e a quantidade de tarefas concorrentes pertencentes ao próprio time e a outros times.

O cálculo de thresholds baseado na distribuição de capacidades não considera concorrência. Ele está baseado no cálculo proposto em [Scerri et al. 2005] e é calculado em função do número de agentes no sistema, da distribuição de capacidades e do número de tarefas da classe presente no time que realiza o cálculo. Em geral, um cálculo de thresholds baseado na distribuição de capacidades não é suficiente para o protocolo. No caso de agentes com interesses próprios, os agentes não possuem toda a informação global necessária pois os times mantêm privadas certas informações, especialmente o valor do pagamento recebido $Pag(\theta_{i,l})$. Para solucionar este problema foi considerado um cálculo de thresholds baseado em execuções prévias do protocolo. Este cálculo, chamado *Learned Threshold* (Threshold Aprendido), permite que cada time armazene os thresholds mínimos e os máximos utilizados em uma execução do protocolo a fim de utilizá-los em futuras execuções e assim possuir thresholds adequados à concorrência atual. Para isso, os administradores comunicam ao time o threshold utilizado uma vez terminada a contratação. Este cálculo de thresholds pode se realizar somente uma vez e servir para futuras contratações.

Uma descrição mais detalhada dos algoritmos para o cálculo de thresholds pode ser encontrada em [Polar 2007].

4. Avaliação Experimental

Para a realização dos experimentos com o protocolo proposto, foi implementada uma tarefa de transporte de mercadorias [Sandholm 1993, Sandholm and Lesser 1995b, McElroy 1989, Fischer et al. 1996]. Trata-se de uma tarefa onde veículos autônomos, pertencentes a diferentes companhias, transportam objetos encarregados por clientes desde uma posição origem até um destino.

O PCS foi comparado com um algoritmo típico de contratação, a contratação gulosa (CG) [Sandholm 1993, Fischer et al. 1996]. Na implementação da contratação gulosa realizada, um agente encarregado de uma tarefa pede informação a todos os agentes no sistema, a fim de calcular a capacidade de cada agente. Logo o administrador vai negociando com cada agente por ordem de capacidade até algum agente aceitar a tarefa. A contratacão gulosa obtém resultados próximos ao ótimo.

O PCS, a tarefa de transporte e o algoritmo de contratação gulosa foram implementados em um simulador multi-robô chamado MuRoS [Chaimowicz et al. 2002]. O PCS foi testado extensivamente a fim de medir três parâmetros importantes dentro do problema da contratação em swarms: a escalabilidade, a pareto-eficiência e a justiça na concorrência. A justiça é um dos fatores que permitem a um protocolo possuir racionalidade individual. Os resultados são apresentados nas Figuras 2 e 3.

A Figura 2(a) mostra a quantidade de mensagens utilizadas para contratar os agentes para realizar 50 e 150 tarefas. Nos testes foram usados 5 times, um custo por distância (distância do veículo até a tarefa) máximo de 1 e mínimo de 0.5 unidades monetárias. Um pagamento por tarefa máximo de 1750 e mínimo de 250 unidades, uma espera máxima de 250 e mínima de 240 passos, um incremento de aluguel de 50 unidades e um tamanho de cluster de 50 unidades quadradas. Um cluster é a posição física de uma classe de tarefas, um grid no espaço de operação das companhias de transporte. Além disso, foi utilizado um decremento de threshold de 0.03 cada 45 passos. Os testes com 50 tarefas agruparam as tarefas em 10 clusters e ocuparam-se 50 clusters nos testes com 150 tarefas. No eixo x estão numeradas as quantidades de agentes envolvidos no experimento e no eixo y as quantidades de mensagens utilizadas. Cada ponto na figura é a média de 10 execuções. Ambos algoritmos utilizaram os mesmos conjuntos de veículos e tarefas em cada execução.

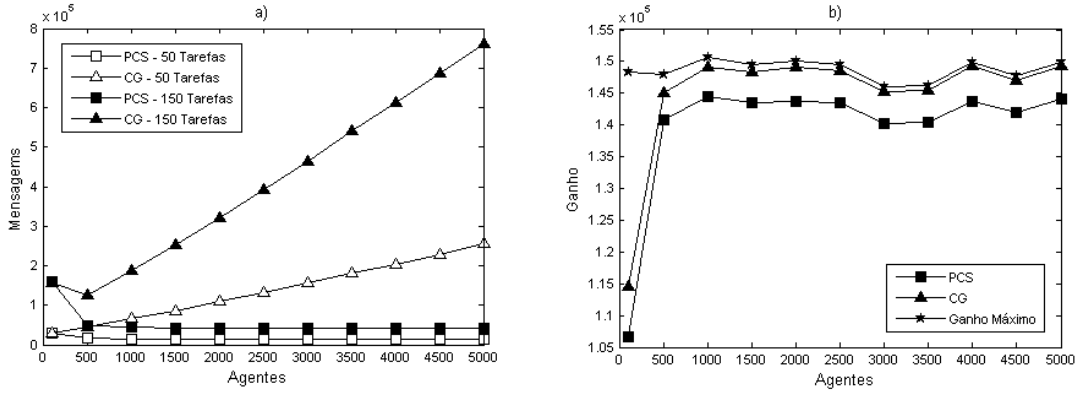

Figura 2. (a) Número de mensagens para 50 e 150 tarefas. (b) Ganho para 150 tarefas.

A Figura 2(a) mostra uma quantidade de mensagens muito inferior para o PCS. A quantidade de mensagens no PCS mantém-se constante ao aumentar a quantidade de agentes no sistema. A quantidade de mensagens na contratação gulosa aumenta linearmente em função da quantidade de agentes. Observa-se também uma diminuição inicial na quantidade de mensagens. O motivo dessa diminuição inicial é que, quando a quantidade de agentes é pequena em relação à quantidade de tarefas, os tokens percorrem o conjunto de veículos por mais tempo antes de encontrar um agente com o que possam negociar. Isso torna a contratação mais complicada.

A Figura 2(b) mostra o ganho obtido nos testes da figura 2(a) para 150 tarefas juntamente com o ganho máximo (soma do pagamentos das tarefas). Como se esperava o ganho é maior para a contratação gulosa. O PCS mantém também um ganho próximo do ótimo. Ambos algoritmos aproximam-se cada vez mais ao ganho máximo. Estes resultados permitem perceber que o PCS é também pareto-eficiente, dado que a quantidade de possíveis melhorias nos contratos (área entre as curvas do PCS e do ganho máximo na figura 2(b)) é pequeno em relação à quantidade total de possíveis soluções.

Os experimentos também determinaram a justiça no PCS com base em dois critérios: concorrência justa e preços justos. Dentro do conjunto de experimentos que medem a concorrência justa, realizaram-se testes que analisaram a justiça de thresholds globais dentro do PCS, um cálculo de thresholds que leva em conta tarefas de outros times como em uma alocação entre agentes cooperativos. Thresholds calculados dessa forma permitiriam diminuir a quantidade de mensagens, mas poderiam tornar o protocolo pouco justo.

Na figura 3(a) mostra-se a capacidade média utilizada pelos times para 3 relações de veículos e tarefas ($\mid E \mid / \mid \Theta \mid$). Cada barra na figura é a média de 10 execuções. No eixo y representa-se a capacidade média utilizada por cada time para contratar todo o conjunto de tarefas em cada execução do protocolo. Utilizou-se 3 clusters e 7 times onde times com maior índice possuem pagamentos médios maiores e estão representados no eixo x da figura 3. O pagamento por tarefa máximo foi de 2500 e o mínimo foi 250 unidades. Os outros parâmetros foram os mesmos do teste de escalabilidade.

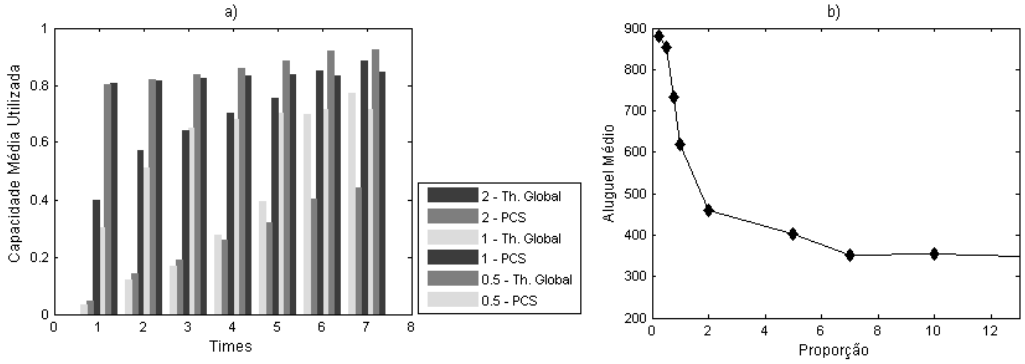

Figura 3. (a) Concorrência Justa no PCS. (b) Aluguel médio para diferentes proporções de tarefas e agentes.

Na Figura 3(a) os times com um pagamento médio maior utilizaram uma capacidade média maior para contratar agentes, o que mostra uma concorrência justa dentro do PCS. Na figura 3(a) também mostra-se a capacidade média utilizada pelos times com um

cálculo de thresholds global. Neste caso, ainda que os times com um pagamento médio maior também utilizaram uma capacidade média maior, a capacidade média utilizada é bem inferior comparada com a capacidade média utilizada dentro do PCS. Portanto, a utilização de thresholds globais não produz uma concorrência justa que outorgue um ganho maior a times que se prepararam melhor para concorrer e em muitos casos diminui a capacidade média utilizada por todos os times.

Agentes podem atuar racionalmente considerando como preço justo um preço determinado pela oferta e demanda e dessa maneira evitar conflitos devido a diferenças em valorizações. Para avaliar se o preço é determinado pela oferta e demanda dentro do PCS, foram executados uma série de experimentos que calcularam o aluguel médio pago pela realização das tarefas em diferentes proporções de agentes e tarefas. Na Figura 3(b) mostra-se o aluguel médio pago para a execução de todas as tarefas presentes no sistema para diferentes proporções ($\mid E \mid / \mid \Theta \mid$). Cada ponto na figura é a média de 10 execuções. Utilizaram-se 7 times e os outros parâmetros foram os mesmos que os utilizados nos testes de escalabilidade.

Na figura 3(b), quando a quantidade de tarefas é grande em relação à quantidade de agentes o aluguel médio é maior. O aluguel médio diminui quando a quantidade de tarefas é pequena em relação à quantidade de agentes no sistema. Portanto o preço é determinado pela oferta e a demanda.

5. Conclusões e Trabalhos Futuros

Até agora as pesquisas em contratação de agentes com interesses próprios (*contracting*) propõem protocolos que não são viáveis para swarms devido à grande quantidade de comunicação e processamento que utilizam. O presente trabalho propõe um novo protocolo que permite a contratação entre agentes com interesses próprios em swarms (PCS).O PCS foi testado em uma tarefa de transporte de mercadorias em um simulador multi-robô chamado MuRoS e mostrou ser escalável, pareto-eficiente e justo.

Existem muitas possíveis rotas futuras de pesquisa referentes ao PCS. Por exemplo, o PCS não considera temas como: decomposição e dependência de/entre tarefas, a execução de múltiplas tarefas simultâneas pelos agentes, penalidades, níveis de esforço, etc. Elementos que estão presentes em outros protocolos de contratação e que poderiam ser incluídos dentro do PCS. Também, é possível utilizar o threshold baseado na concorrência como método para corrigir thresholds calculados para alocar tarefas em grandes grupos cooperativos em ambientes onde o cálculo de thresholds é complexo (não se possui informação global, a distribuição de capacidades muda freqüentemente). Além disso, testes do PCS em outras tarefas e em implementações reais podem também ser feitos.

Referências

Babanov, A., Collins, J., and Gini, M. (2003). Asking the right question: Risk and expectation in multi-agent contracting. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 17(4):173–186.

Chaimowicz, L., Campos, M., and Kumar, V. (2002). Dynamic role assignment for cooperative robots. *The IEEE International Conference on Robotics and Automation*, pages 293–298.

Collins, J., Ketter, W., Gini, M., and Mobasher, B. (2002). A multi-agent negotiation testbed for contracting tasks with temporal and precedence constraints. *International Journal of Electronic Commerce*, 7(1):35–57.

Fischer, K., Müller, J. P., and Pischel, M. (1996). Cooperative transportation scheduling: An application domain for dai. *Journal of Applied Artificial Intelligence. Special Issue on Intelligent Agents*, 10(1):1–33.

Jang, M.-W. and Agha, G. (2004). Dynamic agent allocation for large-scale multi-agent applications. In *Proccedings of The International Workshop on Massively Multi-Agent Systems (MMAS'04)*, pages 19–33.

Konolige, K., Fox, D., Ortiz, C., Agno, A., Eriksen, M., Limketkai, B., Ko, J., Morisset, B., Schulz, D., Stewart, B., and Vincent, R. (2004). Centibots: Very large scale distributed robotic teams. In *Proccedings of The International Symposium on Experimental Robotics (ISER)*.

Kraus, S. (1996). An overview of incentive contracting. *Artificial Intelligence*, 83(2):297–346.

McElroy, J. (1989). Communication and cooperation in a distributed automatic guided vehicle system. In *Proccedings of The IEEE Southeastcon ()*, pages 999–1003.

Polar, C. J. D. (2007). Um protocolo para contratação de agentes em grupos de larga escala. Dissertação de Mestrado, Departamento de Ciência da Computação - Instituto de Ciências Exatas - Universidade Federal de Minas Gerais.

Sandholm, T. (1999). Distributed rational decision making. *Multiagent systems: a modern approach to distributed artificial intelligence*, pages 201–258.

Sandholm, T. W. (1993). An implementation of the contract net protocol based on marginal cost calculations. In *Proccedings of The National Conference on Artificial Intelligence (AAAI)*, pages 256–262.

Sandholm, T. W. and Lesser, V. (1995a). Advantages of leveled commitment contracting protocol. In *Proccedings of The National Conference on Artificial Intelligence (AAAI)*, volume 1, pages 126–133.

Sandholm, T. W. and Lesser, V. R. (1995b). Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *Proccedings of The International Conference on Multi-Agent Systems (ICMAS)*, pages 328–335.

Scerri, P., Farinelli, A., Okamoto, S., and Tambe, M. (2005). Allocating tasks in extreme teams. In *Proccedings of The International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 727–734.

Scerri, P., Xu, Y., Liao, E., Justin, L., and Sycara, K. (2004). Scaling teamwork to very large teams. In *Proccedings of The International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 888–895.

Smith, R. (1980). The contract net protocol. *IEEE Transactions on Computers*, C–29(12).

Xu, Y., Scerri, P., Yu, B., Okamoto, S., Lewis, M., and Sycara, K. (2005). An integrated token-based algorithm for scalable coordination. In *Proccedings of The International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 407–414.