CHRISTIAN JORGE DELGADO POLAR

UM PROTOCOLO PARA CONTRATAÇÃO DE AGENTES EM GRUPOS DE LARGA ESCALA

Belo Horizonte
26 de março de 2007

Departamento de Ciência da Computação–ICEx–UFMG
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

UM PROTOCOLO PARA CONTRATAÇÃO DE AGENTES EM GRUPOS DE LARGA ESCALA

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação do Departamento de Ciência da Computação–ICEx–UFMG como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

CHRISTIAN JORGE DELGADO POLAR

Belo Horizonte
26 de março de 2007

Resumo

Sistemas multi-agente têm sido usados na resolução de diversos problemas em áreas que podem variar do comércio eletrônico à robótica móvel. Nesses sistemas, um agente pode necessitar cooperar com outros para atingir um objetivo. Uma forma de cooperação é a realização de contratos pelos quais um agente pode contratar outro que esteja melhor capacitado para realizar uma tarefa. Os contratos são particularmente importantes quando os agentes não compartilham o mesmo objetivo. Nesse caso, um agente poderá encarregar uma tarefa a outro, que não compartilha os mesmos objetivos, mas realizará a tarefa incentivado por uma recompensa. Atualmente, com os avanços da computação massiva, tem surgido a necessidade de construir sistemas compostos por um número cada vez maior de agentes, em quantidades que podem chegar a centenas ou mesmo milhares. Nestes sistemas, a contratação de agentes deve ser escalável. Este trabalho propõe um protocolo escalável para a contratação de agentes com interesses próprios em grupos de larga escala, geralmente chamados swarms. O protocolo permite a formação de contratos mediante um processo que passa por três etapas: uma etapa de descoberta, onde os agentes encontram outros agentes que podem ser contratados, uma etapa de negociação, onde os agentes envolvidos determinam o preço da contratação e uma etapa de execução, onde a tarefa é alocada e o preço do contrato é pago após a execução da tarefa. No protocolo, o uso de limiares dinâmicos facilita a descoberta de agentes e uma mistura de leilões simultâneos permite aos agentes negociar e definir o preço do contrato. O protocolo foi testado em simulações onde times de robôs pertencentes a diferentes companhias são contratados para transportar mercadorias. Os resultados obtidos mostraram que o protocolo proposto é racional, pareto–eficiente, distribuído e justo, características desejáveis em todo protocolo de contratação. Além disso, o protocolo mostrou-se escalável utilizando níveis baixos de comunicação e consumindo poucos recursos computacionais sendo, portanto, adequado para swarms.

Abstract

Multi-Agent Systems have been used in the solution of various problems in different fields, ranging from electronic commerce to mobile robotics. In these systems, individual agents may have to cooperate in order to fulfill their objectives. A common way of cooperation is the execution of contracts, through which an agent may contract a more capacitated one to execute a specific task. Contracts are particularly important when agents have different objectives. In this case, a agent will contract another one that, despite having other objectives, will execute the task receiving a payment for that. With the advances in massive computing systems, there is an increasing need for technologies that deal with very large numbers of agents. In these systems, generally called swarms, the contracting must be done in a scalable way. The present work presents a scalable protocol for contracting in large groups of self-interested agents. The protocol has three basic phases: a discovery phase that searches for adequate agents for contracting, a negotiation phase, in which the contracting price is determined and an execution phase, in which tasks are allocated and agents receive their payment after task execution. The use of dynamic thresholds facilitates agent discovery and a mix of two types of auctions allows agents to negotiate and determine the values to be paid under the contract. The protocol was tested in simulations where teams of robotic vehicles belonging to different companies must transport items from specific targets. Results demonstrate that the proposed protocol is rational, pareto–efficient, distributed and fair, desirable characteristics in any contracting protocol. Moreover, the protocol is scalable and uses low levels of communication and processing, making it adequate for swarms.

Para meus pais:
Juan e María Antonieta

Agradecimentos

Aos meus pais, María Antonieta Polar Silva e Juan Jorge Delgado Ortiz, por uma vida inteira de carinho e dedicação e por ter-me dado a melhor herança, a fé.

Ao professor e orientador Dr. Luiz Chaimowicz, peça fundamental para a realização desta dissertação, pela presença e suporte profissional em cada passo do desenvolvimento deste trabalho, e pelo apoio pessoal oferecido durante minha estadia na cidade de Belo Horizonte. Ao professor e co-orientador Dr. Mário Campos, pelo apoio profissional e interesse constante por meu bem-estar como estudante estrangeiro.

Ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais (PPGCC/UFMG) pela excelência do curso oferecido e pelo esforço constante em oferecer aos alunos a melhor estrutura possível.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo apoio prestado na realização deste curso.

Por fim, estendo meus agradecimentos a todos aqueles que, direta ou indiretamente, me auxiliaram no progresso deste trabalho.

Sumário

Lista de Figuras

Notação

- $e_{n,l}$ (agente): entidade encarregada de realizar as tarefas pertencentes ao time l. Ex. Veículo Autônomo.
- E: conjunto de agentes no sistema.
- SMA: sistema multi-agente.
- E_k (time): conjunto de agentes que pertencem à mesma organização e possuem os mesmos interesses. Ex. Veículos Autônomos de uma companhia de transporte de mercadoria.
- $L = \{l_1, ..., l_k\}$ (swarm): conjunto formado por uma quantidade grande de agentes. Um swarm é formado por vários times de agentes l_k. Ex. Companhias de transporte de mercadoria.
- $\theta_{i,l,c}$ (tarefa): atividade atribuída (contratada por um cliente externo) a um determinado time l. Sendo c a classe da tarefa. Ex. Transporte de Mercadoria de um lugar origem a um lugar destino.
- Θ: conjunto de tarefas no sistema.
- c (classe de Tarefa): Conjunto de tarefas que requerem as mesmas capacidades dos agentes. Um agente poderá ter uma capacidade diferente para cada classe de tarefa. Ex. Uma classe de tarefa pode ser um quadrante se dividimos o espaço onde operam as companhias de transporte de mercadorias em quadrantes. Então cada tarefa de transporte que está dentro de um quadrante pertence a essa classe.
- Θ_c: conjunto de tarefas da classe c.
- $\Theta_{l,c}$: conjunto de tarefas da classe c do time l.
- $Cap(e_{n,l}, \theta_{m,p,c}) \rightarrow [0, 1]$ (capacidade): Qualificação que recebe um agente $e_{n,l}$ de acordo à eficiência com que realiza uma tarefa $\theta_{m,p,c}$ pertencente a uma classe c. Ex. Proximidade à origem da tarefa.
- $Cus(e_{n,l}, \theta_{m,p,c})$ (custo da Tarefa): custo incorrido pela realização da tarefa $\theta_{m,p,c}$ no agente $e_{n,l}$. O Custo é dado em função da capacidade do agente: $Cus(e_{n,l}, \theta_{m,p,c}) = C_l * (1 - Cap(e_{n,l}, \theta_{m,p,c}))$ onde C_l é o maior custo possível para o time l. Ex. Combustível utilizado pelo veículo para a tarefa de transporte.

- $CusME(Cap(e_{n,l}, \theta_{m,p,c}), \theta_{m,p,c})$ (custo menor esperado): menor custo estimado geralmente incorrido pela realização da tarefa $\theta_{m,p,c}$ num agente qualquer com capacidade $Cap(e_{n,l}, \theta_{m,p,c})$.

- $Pag(\theta_{m,l,c})$ (pagamento): benefício que recebe um time l desde o exterior do sistema pela realização de uma tarefa própria $\theta_{m,l,c}$. Ex. Pagamento do cliente por um transporte de mercadoria.

- $Al(e_{n,p}, \theta_{m,l,c})$ (aluguel): benefício que recebe um time p por alugar um agente $e_{n,p}$ para realizar uma tarefa $\theta_{m,l,c}$ de um outro time l. Ex. Aluguel de veículos a uma companhia.

- $Cont(e_{n,p}, \theta_{m,l,c})$ (contrato): Especificações de execução da tarefa $\theta_{m,l,c}$ e aluguel que recebe o agente $e_{n,p}$. Ex. Contratação de um veículo para um transporte de mercadoria.

- $Gl(\theta_{m,p,c})$ (ganho líquido): benefício líquido que recebe um time p por uma tarefa própria $\theta_{m,p,c}$ considerando aluguéis. Normalmente calculado como a diferença entre o pagamento recebido e o aluguel pago. Ex. Pagamento do cliente por um transporte de mercadoria considerando o aluguel do veículo.

- PCSMAGE: Problema da Contratação em Sistemas Multi-Agente de Grande Escala.

- PCS: Protocolo para Contratação em Swarms.

Capítulo 1

Introdução

O incremento da conectividade e da capacidade de processamento dos dispositivos computacionais têm permitido o início da era massiva da computação. Espera-se, num futuro próximo, cenários onde: milhões de dispositivos eletrônicos com algum poder de computação sejam conectados uns aos outros formando redes complexas extremamente dinâmicas que precisam de um comportamento coerente, inúmeros computadores interconectados via Internet estejam preparados para as mais diversas atividades e para os usos mais diversos do ser humano, grandes grupos de robôs estejam recebendo ordens do ser humano para executar tarefas que não poderiam ser realizadas por um pequeno grupo de indivíduos, milhares de sensores conectados em rede estejam obtendo do ambiente dados úteis para o ser humano numa variedade impressionante de atividades.

Tecnologias preparadas para atuar em ambientes compostos por quantidades extremas de indivíduos serão de grande utilidade para esta era de tecnologia massiva à que o homem está atualmente enfrentado. Os sistemas multi-agente (SMA), que têm servido como uma ferramenta de projeto e implementação para sistemas compostos por um conjunto de indivíduos, agora também estão começando a ser utilizados para projetar sistemas compostos por centenas ou milhares de indivíduos. Apesar disso, muitos dos conceitos desenvolvidos para SMAs até agora não são aplicáveis a sistemas de larga escala. Cada uma das abordagens estudadas em SMA deve ser adaptada para poder ser utilizada em ambientes compostos por grandes grupos de agentes. Dentre essas, está o problema de contratação de agentes com interesses próprios.

Como será discutido nas próximas seções, a presente dissertação propõe um protocolo de contratação para sistemas multi-agente formados por uma grande quantidade de indivíduos que possuem cada um seus próprios interesses.

Este capítulo está organizado da seguinte forma: a motivação é apresentada na Seção 1.1. A Seção 1.2 define o problema abordado. A Seção 1.3 apresenta as contribuições deste trabalho. O capítulo é finalizado com um sumário geral da organização desta dissertação.

1.1 Motivação e Objetivos

Cenários onde grandes grupos de robôs, agentes e humanos realizam tarefas complexas e distribuídas podem ser esperados num horizonte não muito distante. [Scerri et al. (2004); Ishida et al. (2005); Xu et al. (2005b)]. Recentemente, as pesquisas em sistemas multi-agente (sistemas formados por entidades que podem perceber o ambiente e atuar nele) estão concentrando-se em grupos compostos por uma grande quantidade de membros, ou seja, grupos de larga escala compostos por centenas ou milhares de indivíduos. Esses grupos têm recebido diferentes nomes, tais como: *Large Scale Multi-Agent Systems* — LSMAS, *Massively Multi-Agent Systems* — MMAS ou *Swarms*[1].

Em tarefas como: resposta a desastres, resgate, vigilância, exploração espacial, controle de plantas, limpeza, estratégias militares, etc, geralmente um só agente não é útil e um pequeno grupo deles pode não ser suficiente. Nessas situações, precisa-se de uma grande quantidade de agentes para que o sistema possa ser efetivo.

Os sistemas multi-agente podem ser compostos por agentes que compartilham o mesmo objetivo e têm os mesmos interesses. Agentes com interesses comuns podem cooperar se a cooperação ajuda a atingir o objetivo comum (ex. robôs carregando uma caixa). Um sistema desse tipo é chamado de sistema cooperativo. Os SMAs podem ser compostos também por agentes ou times com interesses próprios (ex. agentes que competem por produtos em um site de comércio eletrônico). A interação entre agentes que possuem interesses próprios (*self–interested*) é um problema que tem sido bastante estudado na área de sistemas multi-agente. SMAs desse tipo são chamados de sistemas competitivos quando cada agente aumenta seu benefício se e somente se outro agente diminui seu benefício. Nos sistemas competitivos muitas vezes os agentes mantêm como privadas certas informações para poder concorrer sem desvantagem (Ex. custos para realizar tarefas, pagamentos recebidos pelas tarefas, etc), enquanto outras informações permanecem como públicas e conhecidas por todos os agentes (ex. quantidade de agentes e tarefas no sistema).

Existem também sistemas compostos por agentes cooperativos com interesses próprios, onde os agentes apesar de ter interesses individuais cooperam para poder atingir melhor os objetivos próprios. Nesses sistemas os agentes as vezes cooperam e as vezes concorrem. Eles, bem como os agentes totalmente competitivos, mantêm privadas certas informações como os custos e pagamentos. Uma forma de interação entre agentes cooperativos com interesses próprios é a realização de "contratos"(*Multi-Agent Contracting* [2]) para a execução de tarefas [Kraus (1996); Sandholm e Lesser (1995a); Collins et al. (2002); Fischer et al. (1996); Babanov et al. (2003)]. Basicamente, um agente contrata outro mais capacitado para realizar uma

[1] Nesta dissertação será utilizado o termo *swarm* (enxame), termo que mostra intuitivamente as propriedades que os sistemas deste tipo apresentam fazendo referência aos enxames naturais compostos por um grande número de indivíduos.

[2] Algumas vezes o termo *contracting* ou contratação tem sido utilizado para referir-se somente à contratação entre agentes com interesses comuns como em [Smith (1980a)]. Nesta dissertação o termo será utilizado tanto para referir-se à contratação entre agentes com interesses comuns quanto à contratação entre agentes com interesses próprios como geralmente é utilizado na literatura.

determinada tarefa [Kraus (2001)]. Como os agentes não compartilham os mesmos objetivos o agente mais capacitado pode não se beneficiar em nada ao realizar a tarefa. Portanto, os agentes que precisam alocar tarefas em outros devem poder motivar estes últimos [Kraus (1996)]. Os agentes podem ser motivados pela promessa de algum tipo de recompensa pela realização da tarefa. Tanto a recompensa como as especificações para a realização da tarefa são determinadas num contrato.

Da mesma forma, os swarms podem também ser compostos por times de agentes que possuem interesses próprios, neles também se faz necessário o uso de contratos para poder motivar agentes mais capacitados. Um grupo de sensores móveis onde uma ou mais organizações com objetivos diferentes precisam perceber diferentes tipos de informação (prevenção de desastres, estudos climáticos, estudos biológicos, etc), um grupo composto por muitos robôs que realizam tarefas de limpeza e um outro grupo que realiza tarefas de vigilância, ambos pertencentes a diferentes organizações, um grande grupo de veículos autônomos de transporte onde cada veículo e tarefa de transporte pertencem a uma empresa (sub-time) determinada, um grupo composto por muitos robôs e/ou humanos realizando atividades de exploração (mineral, espacial) onde cada agente representa os interesses de uma determinada organização (sub-time) são exemplos de swarms que necessitam alguma forma de negociação e estabelecimento de contratos para conseguir que agentes com interesses próprios possam cooperar.

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. Até agora, as pesquisas em *contracting* entre agentes com interesses próprios propõem métodos que não estão preparados para swarms. Elas utilizam algoritmos que não são aplicáveis a swarms devido à grande quantidade de comunicação e recursos computacionais requeridos. Em geral, até agora, não existem algoritmos ou protocolos que permitam uma eficiente interação e cooperação entre agentes com objetivos próprios num swarm.

Portanto, o presente trabalho tem por objetivo implementar um novo protocolo que possibilite a cooperação e negociação por meio da realização de contratos entre times com interesses próprios num swarm de agentes. O protocolo adapta as 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)] a fim de facilitar as três etapas envolvidas em toda contratação de agentes [Tesfatsion e Judd (2006)]: a busca de agentes capacitados para realizar as tarefas (*Discovery*), a negociação (*Dealing*) e a execução dos términos dos contratos (*Exchanging*) que inclui a execução de tarefas e pagamento de recompensas.

1.2 Definição do Problema

Em um ambiente composto por um swarm formado por vários times de agentes, onde cada time possui um interesse diferente e está encarregado de um conjunto de tarefas, o *Problema da Contratação em Sistemas Multi-agente de Grande Escala* (PCSMAGE) abordado nesta dissertação consiste em encontrar um conjunto de contratos para alocar tarefas em agentes.

A maneira de encontrar esse conjunto de contratos deve ser eficiente, de forma que sejam utilizados níveis baixos de comunicação e processamento, e deve permitir maximizar o ganho total obtido por todo time. Esse ganho é dado em função da recompensa recebida pelo time e dos custos incorridos pela realização de todas as tarefas, além dos pagamentos efetuados entre times por causa dos contratos.

Preferivelmente, o método utilizado para encontrar esse conjunto de contratos deve ser simples, eficiente, distribuído e individualmente racional.

1.3 Contribuições

A principal contribuição do presente trabalho é um protocolo para a realização de contratos entre times de agentes com interesses próprios num swarm. A realização de contratos permite que os times possam colaborar e beneficiar-se mutuamente.

Outra contribuição do trabalho é um método para alocar tarefas em swarms totalmente cooperativos baseado em aprendizagem. Isso, dado que o protocolo permite também a contratação de tarefas dentro de times que possuem o mesmo interesse.

1.4 Organização da Dissertação

O restante desta dissertação está organizado da seguinte maneira: o Capítulo 2 apresenta alguns fundamentos básicos e faz referência aos trabalhos relacionados, detalhando os pontos em comum com o presente trabalho. O Capítulo 3 detalha o protocolo proposto, os mecanismos de descoberta de agentes e de negociação para a realização de contratos em swarms. O Capítulo 4 apresenta os resultados dos testes realizados com a implementação atual do protocolo: primeiramente são descritas a metodologia dos experimentos e em seguida a implementação realizada. Por fim, o último capítulo apresenta as conclusões e as perspectivas de trabalhos futuros.

Capítulo 2

Fundamentos e Trabalhos Relacionados

Um agente é um sistema de computação situado em algum ambiente que é capaz de agir de forma autônoma nesse ambiente a fim de conseguir seus objetivos de projeto [Wooldridge (2002)]. Os agentes podem resolver problemas em conjunto quando estão dentro de um sistema multi-agente (SMA), um sistema composto por múltiplos agentes. Nesses sistemas os agentes devem cooperar quando a solução de um problema está além das capacidades individuais de cada um. Uma forma de cooperação é a realocação de tarefas. Um agente pode não estar suficientemente capacitado para realizar uma tarefa que lhe foi encarregada e, nesse caso, pode tratar de alocar a tarefa para outro agente que esteja mais capacitado.

Quando surge a necessidade de realocar tarefas, pode surgir também a necessidade de especificar as condições da cooperação: a forma como deve ser realizada a tarefa, a recompensa que receberá o novo agente ou a forma como ele pode se eximir do compromisso (*decommitting*), entre outras. O elemento que permite a especificação das condições da cooperação entre agentes é o *contrato*. Portanto, um agente pode contratar outro para a execução de uma tarefa.

A contratação pode se realizar entre agentes com interesses comuns ou entre agentes com interesses individuais. Na segunda metade da década de 80 as pesquisas em inteligência artificial distribuída (DAI – *Distributed Artificial Intelligence*) começaram a tomar uma nova direção. Elas começaram a referir-se não somente a agentes totalmente cooperativos, como tinha sido desde o início dos estudos em SMAs, mas também a agentes com interesses próprios (*self–interested agents*). A proliferação de projetos na implementação de agentes inteligentes obrigou a se pensar na interação entre agentes projetados por diferentes organizações. Eles poderiam não somente possuir interesses próprios mas inclusive ser competitivos.

O presente capítulo descreve trabalhos e conceitos relativos à contratação de agentes (*contracting*) tanto para agentes totalmente cooperativos (Seção 2.2.2) como para agentes com interesses próprios (Seção 2.2.3). Além disso, apresenta-se uma breve descrição sobre os conceitos relacionados com swarms de agentes (Seção 2.1). Contratação de agentes, agentes com interesses próprios e swarms são os três elementos mais importantes dentro do PCSMAGE.

2.1 Swarms

Um grupo de formigas procurando alimento, um bando de aves voando juntas, um grande cardume de peixes nadando e virando repentinamente (Figura 2.1), um enxame de abelhas construindo uma colméia; todos se comportando como uma unidade perfeitamente coordenada. Quem governa estes grupos? Como planejam? Como se coordenam? Como preservam o equilíbrio? São perguntas que fascinam qualquer um, desde naturalistas até artistas. Grupos deste tipo são geralmente chamados *swarms* [1].

Figura 2.1: Um cardume de peixes.

Apesar da grande quantidade de indivíduos, os swarms apresentam-se robustos, flexíveis e auto-organizados [Bonabeau et al. (1999)]. Em anos recentes, diversos esforços foram realizados para modelar o comportamento destes grupos, obter um conhecimento mais profundo sobre eles e utilizar este conhecimento como exemplo para inspirar algoritmos que utilizem as propriedades mais interessantes dos swarms para resolver diversos problemas. Estudos deste tipo podem ser agrupados sob o termo *swarm intelligence*. Os resultados dos estudos nessa área estão sendo aplicados em diversas outras áreas. Por exemplo: otimização, sistemas multi-agente, robótica, telecomunicações, padrões de tráfego, sistemas de transporte, aplicações militares, etc.

Uma visão global de um swarm sugere que n agentes estão cooperando para atingir algum objetivo. No entanto, cada agente parece ter uma agenda própria sem se preocupar em coordenar-se com o resto dos agentes no swarm. Essa característica é a chamada *inteligência coletiva* que surge do comportamento simples de cada agente. Em todo swarm não existe um ponto central de controle e cada agente comporta-se seguindo um conjunto simples de regras. Uma ave voando em grupo somente se preocupa em manter-se perto do resto do bando evitando colidir com alguma delas. Ela somente pode perceber as aves mais próximas, e não recebe ordens de alguma outra ave, ou seja, não existe uma ave líder.

Cada peixe que nada dentro de um cardume somente pode perceber os que estão mais próximos dele. Eles possuem somente uma visão local do ambiente. Devido ao grande número

[1] Apesar de existirem diferentes formas coletivas para referenciar esses grupos (cardumes, manadas, bandos, etc.) o termo *swarm* (exame) tem sido usado na literatura, principalmente nas áreas de sistemas multi-agente e robótica.

de indivíduos e à simplicidade que possuem, cada agente dentro do swarm não pode possuir informação de todos outros agentes. Cada agente possui conhecimento de um grupo muito reduzido de vizinhos e ainda assim podem conseguir uma coordenação do grupo inteiro e resolver problemas complexos (Ex. encontrar a rota mais curta entre o ninho e uma fonte de comida). Esta é uma característica que permite aos swarms reagir rápida e flexivelmente a mudanças no ambiente.

A grande quantidade de indivíduos no grupo e a distribuição que apresentam permite que a falha de um deles não represente um problema para atingir o objetivo comum. Uma formiga que se perde não impede que o resto encontre a fonte de comida. Existem muitos agentes que podem realizar a mesma tarefa. Eles apresentam características similares, o que permite que um agente possa ser substituído por outro quando este falha. Devido a isso, os agentes num swarm são anônimos. Não existe a necessidade de diferenciar dois indivíduos que possuem as mesmas características e podem realizar as mesmas atividades. Eles são tratados mais como um grupo que como indivíduos particulares.

Uma comunidade cada vez maior de pesquisadores tem estudado recentemente novas maneiras de aplicar a inteligência dos swarms em diversas tarefas [Bonabeau e Théraulaz (2000)]. O estudo dos mecanismos de busca de alimento das formigas levou ao projeto de métodos inovadores para conduzir o tráfego de rede em sistemas de telecomunicações. A interação cooperativa de formigas que trabalham para transportar um grande pedaço de alimento pode também inspirar algoritmos mais eficazes para a robótica. A maneira como os insetos agrupam os mortos na colônia e classificam as larvas pode ajudar na análise de dados em atividades bancárias. A divisão do trabalho entre abelhas poderia ajudar na organização em plantas de linha de montagem em fábricas.

Uma das áreas onde os conceitos de swarms podem ser muito aplicados é a robótica. Os swarms dão a possibilidade de melhorar a coordenação de robôs, diminuir os requerimentos de comunicação, aumentar a tolerância a falhas e diminuir os custos e a complexidade de sistemas tradicionais de múltiplos robôs. Diversos trabalhos têm abordado o problema de swarms em robótica [Chaimowicz et al. (2005); Correll et al. (2006); Hsieh e Kumar (2006); Konolige et al. (2004); McLurkin e Smith (2004); Pimenta et al. (2006)]. Em [Chaimowicz et al. (2005)], por exemplo, é apresentado um algoritmo escalável e flexível que permite controlar um swarm de robôs utilizando funções implícitas (Figura 2.2). No artigo, controla-se o movimento dos robôs com regras simples a fim de que sigam as curvas geradas pelas funções fazendo uso de uma modificação da técnica de descida do gradiente. Em [Konolige et al. (2004)] é apresentado um projeto chamado *Centibots* (Figura 2.3) que tem o objetivo de projetar, implementar e demonstrar um framework para a coordenação de times de robôs de grande escala numa tarefa de busca e resgate. No projeto Centibots considerou-se uma população homogênea de robôs que realizavam tarefas num ambiente com comunicação limitada e pouco confiável. Conseqüentemente, os mecanismos de coordenação tiveram que enfrentar estas condições.

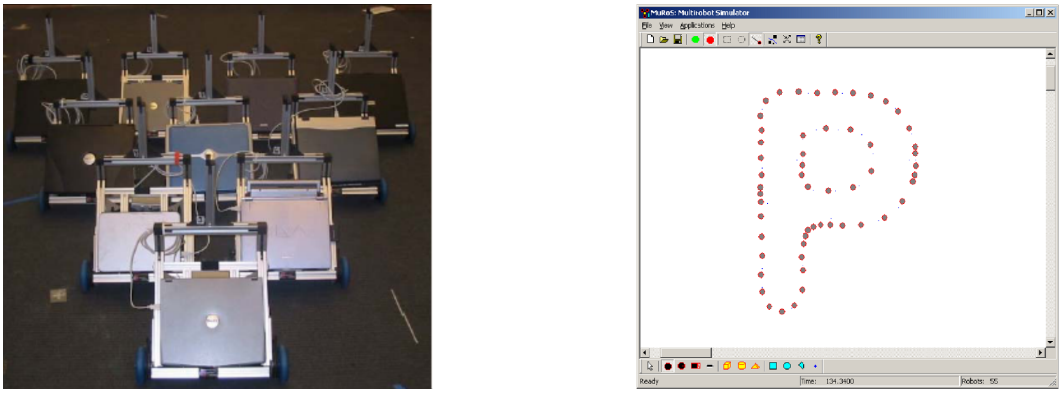

Figura 2.2: 10 robôs ER1 utilizados nos experimentos e uma simulação de robôs formando a letra P. [Chaimowicz et al. (2005)].

Figura 2.3: 100 robôs utilizados no projeto Centibots. [Konolige et al. (2004)].

2.2 Contratação de Agentes

Como mencionado, um agente pode contratar outros para executar uma tarefa quando ele não puder realizá-la, ou mesmo quando ela puder ser mais eficientemente executada por outros agentes [Kraus (2001)]. Pode existir contratação entre agentes que compartilham o mesmo objetivo ou entre agentes que possuem objetivos diferentes. A contratação entre agentes que possuem o mesmo objetivo é mais conhecida como *alocação de tarefas* (*task allocation*) [Gerkey e Mataric (2004)] pois, geralmente, nesse contexto não é necessário o uso de contratos que especifiquem as condições da cooperação. Determinar elementos como: o nível de esforço que um agente compromete-se a utilizar para realizar uma tarefa, a recompensa que receberá o agente ou o valor que pagará a fim de eximir-se do contrato são necessários para sistemas de agentes que possuem interesses próprios mas, geralmente, esses elementos não precisam ser definidos em sistemas de agentes que compartilham o mesmo objetivo. Em times que compartilham o mesmo objetivo, os agentes, sempre que é possível, utilizarão o máximo esforço para realizar as tarefas. A recompensa obtida é sempre o ganho total obtido pelo time na realização da tarefa e, dado que os agentes sempre procuram o benefício de todo o time, não existe a necessidade de penalidades para eximir-se de um compromisso. Em geral, os algoritmos de alocação de tarefas concentram-se principalmente em: escolher as tarefas que

produzem mais ganho, determinar o melhor agente para realizar uma tarefa ou a melhor forma de decompor uma tarefa em sub-tarefas. Algoritmos desse tipo não resolvem problemas como: o cálculo de preços, a partilha do ganho entre times, honestidade, especulação ou penalidades. As seguintes seções tratam os temas de alocação de tarefas e a contratação de agentes com interesses próprios. Além disso, descreve-se o *Contract Net Protocol* – CNP um protocolo muito utilizado na contratação de agentes em geral.

2.2.1 Contract Net Protocol

O Contract Net Protocol – CNP [Smith (1977, 1980a,b); Smith e Davis (1981)] é um protocolo de cooperação onde os agentes podem compartilhar tarefas na resolução de diversos problemas, sendo a sua versão original voltada para os problemas em redes de comunicação. O CNP está inspirado na forma como as companhias realizam o processo de contratação de serviços. Na Figura 2.4, mostra-se o processo executado pelo CNP. Basicamente, ele funciona da seguinte forma [Smith (1980b) pp. 62,63]:

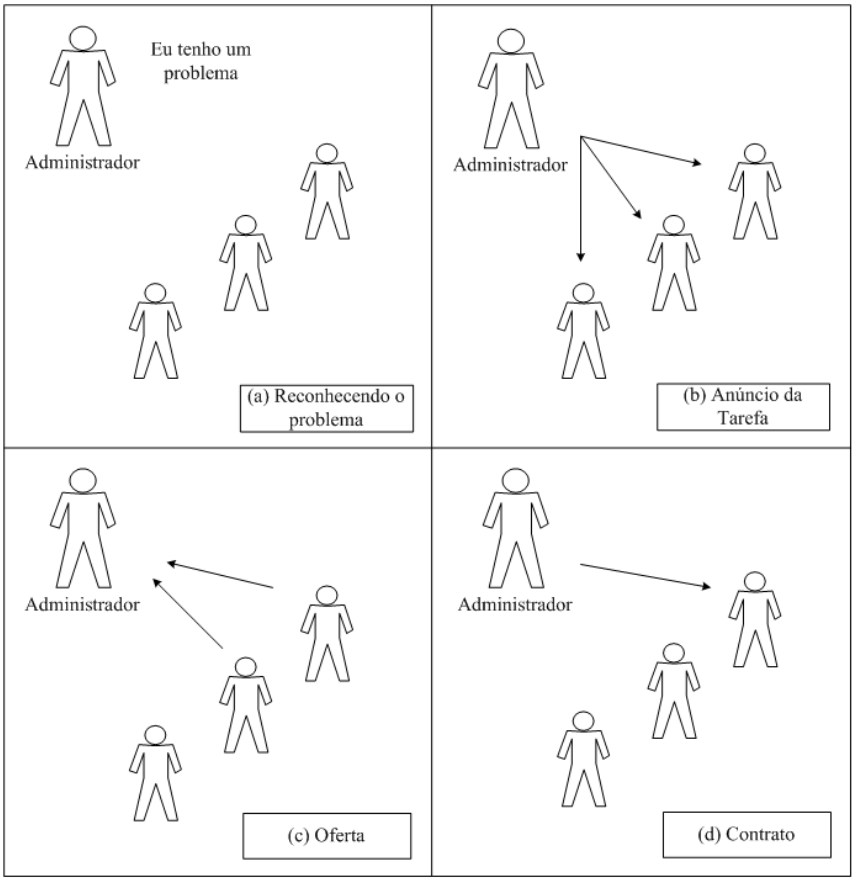

Figura 2.4: Contract Net Protocol.

"Um nó gera um aviso da existência de uma tarefa a outros nós na rede com *anúncios de tarefas* e depois atua como o administrador dessa tarefa por sua duração. Na ausência de informação relacionada às capacidades de outros nós na rede, o administrador é forçado

a uma difusão geral a todos os outros nós. No entanto, se o administrador possui algum conhecimento sobre quais nós na rede podem ser candidatos, ele pode realizar uma difusão limitada justamente para esses nós. Além disso, se o administrador conhece exatamente qual dos outros nós é apropriado, ele pode realizar um anúncio ponto a ponto. Conforme o trabalho no problema progride, muitos anúncios de tarefas serão realizados por vários administradores.

Nós na rede escutam os anúncios de tarefas e os avaliam com base nos recursos de hardware e software que possuem. Um nó que é adequado para uma tarefa submete uma oferta. Uma oferta indica as capacidades do agente que são relevantes para a execução da tarefa anunciada. O administrador pode receber muitas ofertas em resposta a um simples anúncio de tarefa. Baseado na informação das ofertas ele pode escolher os nós mais adequados para realizar a tarefa. A escolha é comunicada aos nós escolhidos através de uma mensagem de concessão. Os nós escolhidos responsabilizam-se pela execução da tarefa e cada um é chamado o executor (*contractor*) para essa tarefa.

Este processo normal de negociação pode ser simplificado em alguns casos, provocando uma melhora na eficiência do protocolo. Se o administrador conhece exatamente que nó é adequado para a execução da tarefa, um *contrato direto* pode ser concedido. Isso difere do anúncio de tarefas pois nenhum anúncio é realizado e nenhuma oferta é submetida. Somente uma concessão é realizada diretamente. Nesse caso os nós escolhidos podem aceitar ou recusar o contrato.

Finalmente, para tarefas que precisam de informação para ser realizadas, um contrato pode não ser adequado. Nesses casos, uma seqüência de requerimentos-respostas pode ser usada. As mensagens (que ajudam na distribuição de dados) são implementadas como mensagens de requerimento e de informação. As mensagens de requerimento são usadas para encapsular requerimentos diretos de informação quando a contratação não é necessária. As mensagens de informação são usadas para ambos: responder a mensagens de requerimento ou como simples mensagens de transferência de dados".

2.2.2 Alocação de Tarefas

Cada agente num sistema multi-agente pode atuar de forma independente dentro do sistema e ainda assim conseguir completar algumas tarefas em conjunto (limpeza de lixo, vigilância, etc). No entanto o verdadeiro potencial desses sistemas é obtido quando eles fazem uso de coordenação. A coordenação é o processo onde as decisões individuais dos agentes resultam em boas decisões conjuntas do grupo [Vlassis (2003)].

A alocação de tarefas é uma forma de coordenação. Os agentes determinam quem realizará cada uma das atividades evitando duplicação do trabalho e obtendo eficiência na realização de cada tarefa. As pesquisas em alocação de tarefas estudam a melhor forma de atribuir tarefas a agentes, tentando utilizar eficientemente os recursos (agentes) para poder conseguir realizar os objetivos do sistema. A importância da alocação de tarefas está no fato de que se os agentes não determinam uma forma eficiente de dividir o trabalho, também não terão uma forma eficiente de executá-lo.

A alocação de tarefas tem sido amplamente tratada nas áreas da robótica e dos agentes de software. Dentro destas propostas existem algumas que se concentram mais na coordenação e outras que definem melhor um algoritmo específico para alocar tarefas. Nas seguintes seções serão descritas algumas das propostas mais importantes que existem em ambas as áreas e analisadas as características que as tornam adequadas para sua utilização em pequenos ou grandes grupos de agentes.

2.2.2.1 Alocação de Tarefas e Coordenação em Pequenos Grupos

Dentro da robótica uma proposta interessante é a arquitetura Alliance [Parker (1998)]. Ela é uma arquitetura completamente distribuída para a coordenação de grupos de robôs, que foi inicialmente proposta como um sistema de coordenação tolerante a falhas e que se adapta dinamicamente ao ambiente. Para isso, incorpora no comportamento dos robôs dois tipos de motivação: impaciência e conformidade. Motivação que os robôs atingem dependendo do progresso das tarefas e dos estados internos que atualmente possuem. O mecanismo que o Alliance utiliza para atualizar a motivação dos robôs não está baseado numa observação passiva, que seria complicado de implementar fisicamente. O mecanismo está baseado em comunicação explícita: cada robô transmite uma mensagem para o resto do grupo contendo informação sobre a ação que o robô está atualmente executando. Alliance tem obtido bons resultados na coordenação de grupos de robôs de tamanho pequeno e médio, no entanto o mecanismo de comunicação utilizado, onde cada robô comunica-se continuamente com todo o resto do grupo, faz com que esta proposta seja inviável em swarms.

Alguns algoritmos para realizar alocação de tarefas têm sido inspirados em conceitos de mercado. Um dos primeiros mecanismos para alocação de tarefas baseado explicitamente em conceitos de mercado é o M+ [Botelho e Alami (1999)]. M+ é um protocolo que permite três atividades: alocação de tarefas, reação cooperativa a contingências e execução de tarefas. O mecanismo para alocação de tarefas do M+ permite uma escolha eficiente do robô que executará uma determinada tarefa. Essa escolha está baseada na comparação das ofertas que fazem os robôs frente a uma tarefa. No M+, todas as ofertas são dirigidas para um robô (o primeiro robô que oferta). Esse robô, portanto, recebe $O(n)$ mensagens por cada tarefa, sendo n o número de robôs no sistema. Essa característica e o fato de que cada robô difunde uma mensagem para o resto dos robôs, cada vez que começa ou finaliza uma tarefa, fazem que este método não seja aplicável para grupos de grande escala.

Outra proposta baseada em conceitos de mercado é Murdoch [Gerkey e Mataric (2002)]. Murdoch é um mecanismo para alocação de tarefas baseado em leilão. Ele utiliza cooperação explícita baseada num modelo de comunicação do tipo *publish/subscribe*. Cada vez que uma nova tarefa entra no sistema, um agente (leiloeiro) publica uma mensagem com informação sobre a tarefa. Todos os robôs que têm os recursos necessários para executar a tarefa são notificados (robôs previamente registrados). Depois disso, cada robô notificado realiza uma oferta. O leiloeiro determinará a melhor oferta e a tarefa será atribuída a esse robô. Numa situação onde vários robôs podem estar habilitados a realizar uma tarefa determinada, o

leiloeiro precisaria de $O(n)$ mensagens. Essa característica faz que o Murdoch seja inviável para grupos de grande escala.

Uma proposta que aborda o problema de alocação de tarefas em ambientes dinâmicos é [Chaimowicz et al. (2002)]. Nela apresenta-se um método para atribuir dinamicamente papéis a robôs cooperativos. O dinamismo deste método permite melhorar a adaptação ao ambiente e melhorar o desempenho de cada indivíduo em benefício do grupo, uma característica interessante no caso dos swarms. Essa proposta é testada numa tarefa de transporte de objetos, onde a atribuição de tarefas realiza-se de forma distribuída. Não obstante, o método também sofre de problemas de escalabilidade, uma vez que utiliza comunicação global.

2.2.2.2 Alocação de Tarefas e Coordenação em Swarms

Muitos trabalhos na comunidade de agentes de software abordam diretamente o problema de coordenação escalável [Turner e Jennings (2000); Jang e Agha (2004); Tosic e Agha (2004); Shehory et al. (1998); Xu et al. (2005a)]. Em [Turner e Jennings (2000)] propõe-se um modelo de adaptação e auto-organização para grupos de grande escala, onde se faz uso de uma auto-organização dinâmica baseada em 3 topologias de agentes. Essas topologias permitem que o sistema seja mais ou menos centralizado, dependendo da tarefa atual. Essa característica faz com que o modelo seja mais adaptável a diferentes tipos de tarefas e também permite diminuir o esforço necessário para coordenar o grupo de agentes, melhorando a escalabilidade. No entanto, ainda que a quantidade total de mensagens requeridas para realizar uma tarefa seja diminuída, muitas vezes um agente deve enviar mensagens ao resto do grupo para conseguir terminar uma tarefa. Isso limita a escalabilidade.

Quando um conjunto de agentes de software é distribuído numa rede de computadores, a localização de cada agente na rede pode influir no desempenho deles. Em [Jang e Agha (2004)] propõe-se uma arquitetura, chamada *Adaptive Actor Architecture* (AAA), que permite controlar a sobrecarga de comunicação na rede e a sobrecarga de processamento em nodos particulares. Essa arquitetura move grupos de agentes para diferentes localizações numa rede a fim de diminuir a sobrecarga de nodos ou diminuir a sobrecarga de comunicação. Todo esse processo é realizado de maneira distribuída, propriedade que o torna adequado para sistemas de grande escala. Entretanto, o AAA não é um método de alocação de tarefas. No AAA as tarefas já estão atribuídas e somente melhora-se a localização dos agentes com base nessa atribuição. Um método de alocação de tarefas poderia melhorar o desempenho da rede sem movimentar agentes, isto é, o procedimento inverso.

Outra proposta baseada na comunicação entre os agentes é apresentada em [Tosic e Agha (2004)]. Essa proposta utiliza a topologia de conexão dos agentes para determinar, de forma completamente distribuída, cliques no grupo de comunicação. Cada clique representa um possível agrupamento já que um agente preferirá formar um grupo com agentes com os quais pode se comunicar. Nesta arquitetura cada agente escolhe se unir a um grupo (clique) que tenha os recursos necessários para executar uma tarefa. Depois desse processo as tarefas ficarão atribuídas a cada grupo, no entanto, a atribuição de tarefas a agentes particulares não

é realizada. Uma característica importante dessa proposta é a formação de grupos baseados nas tarefas a fim de melhorar a sua alocação.

Em [Shehory et al. (1998)], propõe-se um modelo baseado em estados contínuos e movimento de partículas. Trata-se de um mecanismo onde todos os elementos lógicos que participam da alocação de tarefas são representados por elementos físicos em um sistema de campos potenciais. Neste modelo os agentes e as tarefas são representados por partículas. Os agentes são atraídos pelas tarefas e repelidos por outros agentes e obstáculos. Além disso, o movimento dos agentes representa o progresso de uma tarefa. A eficiência das propostas baseadas em detecção local de objetos [Chaimowicz et al. (2002); Shehory et al. (1998)] está fortemente determinada pelo espaço de interação ou influência dos elementos que compõem estes sistemas. Se o campo de interação é pequeno a complexidade diminui, mas a satisfação de objetivos é pobre. Se o campo de interação é maior, a complexidade aumenta devido à interação com uma maior quantidade de agentes.

Métodos inspirados em swarms naturais foram propostos para conseguir uma coordenação escalável [Ferreira et al. (2005); Cicirello e Smith (2001); Valckenaers et al. (2001)]. Em [Ferreira et al. (2005)] por exemplo, cada tarefa possui um estímulo, que é atualizado de acordo com a execução de tarefas relacionadas e depende de fatores como a qualidade e custo da tarefa. Outro elemento importante nesta proposta é um valor limite de resposta (limiar) que cada agente possui para cada tarefa. Esse limiar é atualizado dependendo de elementos como a idade do agente ou a idade da tarefa. A proposta está principalmente orientada a conseguir uma adaptação eficiente a mudanças na natureza das tarefas e do ambiente. Em geral, os métodos baseados em swarms podem ser interessantes para conseguir uma coordenação escalável, isso devido principalmente à natureza distribuída e local que possuem. Porém, geralmente essas técnicas consideram o grupo de agentes como homogêneos o que nem sempre acontece em grupos de grande escala. Trabalhos deste tipo tratam principalmente os temas da adaptabilidade onde os agentes mudam seu comportamento dependendo das tarefas presentes no sistema, e da escolha da melhor tarefa a ser executada em cada momento.

2.2.2.3 Alocação de Tarefas baseada em Tokens

Uma série de trabalhos recentes [Scerri et al. (2005, 2004); Xu et al. (2004, 2005a,b)] faz uso de pacotes de informação (tokens) a fim de transportar tarefas, informações e recursos entre os agentes. Em geral, estas propostas obtêm bons resultados em termos de escalabilidade. Nestes trabalhos as tarefas são encapsuladas em tokens que se movem pelos nós de uma rede, onde cada nó representa um agente que pode executar uma tarefa. Em [Xu et al. (2005b)], por exemplo, faz-se uso de redes sociais [Gaston e desJardins (2005)], para transportar tarefas, informação e recursos em tokens. Nessa proposta um método que integra algoritmos para alocação de tarefas [Scerri et al. (2005)], alocação de recursos [Xu et al. (2005a)] e compartilhamento de informação [Xu et al. (2004)], é utilizado para gerar modelos de rotas de tokens. Os modelos estão baseados na relação entre tokens, que é modelada na teoria de Processos de Decisão de Markov Parcialmente Observáveis (POMDP). Um modelo desse tipo é gerado

progressivamente em cada nó da rede, sendo que o mesmo permite que o agente possa dirigir os tokens para o agente mais adequado.

Em Scerri et al. (2005) é apresentado um algoritmo chamado LA–DCOP (*Low–communication Approximate* DCOP [2]) para alocar tarefas em times *extremos*. Nesses times, existe uma grande quantidade de agentes que podem realizar múltiplas tarefas ao mesmo tempo, muitos agentes podem realizar a mesma tarefa, o ambiente é extremamente dinâmico (as características do grupo de agentes e das tarefas muda freqüentemente) e existem restrições para a execução das tarefas (como por exemplo, execução simultânea).

A proposta contém três idéias novas: Primeiro, o LA–DCOP melhora a eficiência não encontrando o ganho ótimo, mas sim um ganho aproximado por meio de um limiar calculado com base na informação probabilística disponível. O limiar representa a mínima capacidade que deve ter um agente para poder realizar uma tarefa. Uma segunda idéia é que LA–DCOP é o primeiro trabalho que encapsula tarefas em tokens a fim de obter acesso exclusivo para elas. Dessa forma, uma tarefa somente pode ser realizada por um agente em um dado momento. A terceira idéia do LA–DCOP é o uso de tokens potenciais que agrupam tarefas interdependentes a fim de que sejam executadas simultaneamente.

No LA–DCOP cada tarefa é encapsulada num token $\Delta_i =< \theta_j, Th, path >$ onde θ_j é a tarefa encapsulada, Th é o limiar calculado para o token e $path$ é a rota já percorrida pelo token. Δ_i percorre o conjunto de agentes procurando um agente e_k que esteja capacitado. $Cap(e_k, \Delta.\theta_j)$ determina a capacidade do agente e_k para realizar a tarefa θ_j do token Δ. Para saber se um agente está capacitado para realizar a tarefa faz-se uso de Th. Th é calculado com base na informação global (número de agentes no sistema, número de tarefas e distribuição de capacidades). Um agente e_k pode realizar uma tarefa θ_j se possui uma capacidade $Cap(e_k, \Delta.\theta_j)$ maior ou igual do que Th.

O algoritmo 1 mostra o comportamento que possui todo agente no LA–DCOP. Cada vez que um agente recebe uma mensagem, se a mensagem é um token, ele deve decidir se é melhor para o time que ele fique com o token ou é melhor que passe o token a um outro agente (algoritmo 1 linha 9).

Se o agente aceita o token, determina que subconjunto dos tokens aceitos maximizam seu ganho dentro dos recursos que possui. Esse cálculo é realizado pela função *MaxCap* (algoritmo 1 linha 17). O subconjunto de tokens que não maximizam o ganho são passados a outros agentes. Cada token mantém uma lista de agentes já visitados para evitar passar novamente ao mesmo agente.

Este algoritmo também permite que os agentes administrem tokens potenciais. Os tokens potenciais são utilizados quando existem tarefas que precisam ser executadas com algum tipo de dependência. LA–DCOP está preparado para executar tarefas que precisam de execução simultânea e pode ser adaptado para outro tipo de dependência. Um conjunto de tokens potenciais é criado para cada tarefa que precisa de execução simultânea. Os tokens potenciais

[2] Em um problema de otimização distribuída com restrições (*Distributed Constraint Optimization Problem* – DCOP) agentes cooperativos, cada um em controle de uma ou mais variáveis, trabalham juntos para otimizar um conjunto de restrições existentes sobre essas variáveis.

são tratados da mesma maneira como qualquer token comum, com a exceção de que quando um agente aceita um token potencial, espera até que exista pelo menos um agente que tenha aceito um dos tokens potenciais para cada uma das tarefas que precisam uma execução simultânea.

Algoritmo 1 LA–DCOP

```
1: T ← φ, PT ← φ
2: while true do
3:    msg ← ObterMensagem()
4:    if msg is token then
5:       Δ ← msg
6:       if Δ.Th = NULL then
7:          Δ.Th ← CalcLimiar(Δ)
8:       end if
9:       if Δ.Th < Cap(e_k, Δ.θ_j) then
10:         if Δ is potencial then
11:            PT ← PT ∪ Δ
12:            EnviarMensagem("retido", proprietrio(Δ))
13:         else
14:            T ← T ∪ Δ
15:         end if
16:         if Σ_{Δ_i∈T} Recursos(Δ_i) ≥ e_k.recursos then
17:            out ← T − MaxCap(T)
18:            for each Δ_i ∈ out do
19:               if Δ_i is potencial then
20:                  EnviarMensagem("liberado", proprietrio(Δ_i))
21:                  PassarToken(novoToken(Δ_i, potencial))
22:               else
23:                  PassarToken(novoToken(Δ_i))
24:                  T ← T − out
25:               end if
26:            end for
27:         end if
28:      else
29:         PassarToken(Δ)
30:      end if
31:   else
32:      if msg is lock Δ_h E* then
33:         if Δ_h ∈ PT then
34:            PT ← PT − Δ_h
35:            T ← T ∪ Δ_h
36:         else
37:            ∀e_m ∈ E * EnviarMensagem("liberado", e_m)
38:         end if
39:      else
40:         if msg is "liberado"Δ_h then
41:            PT ← PT − Δ_h
42:         end if
43:      end if
44:   end if
45: end while
```

Quando um agente aceita um token potencial o armazena dentro de um conjunto de tokens potenciais PT (algoritmo 1 linha 11) e informa ao proprietário que o token foi aceito. O proprietário é o agente que cria todos os tokens para as tarefas que precisam de uma execução simultânea e se encarrega de coordenar a alocação. No caso em que um token potencial seja excluído por falta de recursos, o agente deve comunicar a liberação do token ao proprietário (algoritmo 1 linha 20). As últimas linhas no algoritmo (algoritmo 1 linha 32–43) determinam o processo que se segue quando já existe pelo menos um agente para cada tarefa que precisa ser executada simultaneamente. O token potencial passa a fazer parte dos tokens aceitos pelo agente e com isso é possível começar a execução da tarefa.

O protocolo proposto nesta dissertação utiliza uma versão adaptada deste algoritmo a fim de descobrir agentes capacitados. Como será detalhado na Seção 3.7, o cálculo de um dos limiares utilizados pelo protocolo está baseado no cálculo de limiares apresentado nessa proposta.

2.2.3 Contratação entre Agentes com Interesses Próprios

Quando os agentes possuem interesses próprios, a dificuldade na contratação aumenta. Para realizar uma contratação de agentes com interesses próprios, não somente é necessário encontrar uma eficiente alocação de tarefas e agentes mas também solucionar problemas como: partilha do ganho na cooperação, níveis de esforço, penalidades, informação privada, honestidade, especulação, etc. Devido a isso, em toda contratação entre agentes com interesses próprios podem surgir conflitos de interesses. Para evitar conflitos na contratação, devem ser desenvolvidos protocolos que evitem elementos como a falta de honestidade ou a revelação de informação privada, e que promovam a determinação clara das regras do jogo através de um contrato onde se especifiquem aspectos como: o nível de esforço que deve ser empregado para a realização de uma tarefa, as penalidades que serão aplicadas em caso da não execução de tarefas e as recompensas pela realização das tarefas.

Obviamente, neste contexto, um protocolo como o CNP já não é suficiente. O CNP foi inicialmente projetado para ambientes totalmente cooperativos, mas existem na literatura algumas extensões do CNP para ambientes compostos por agentes com interesse próprios. Uma das primeiras extensões do CNP foi apresentada em [Sandholm (1993)]. Essa proposta provê uma formalização dos processos de oferta e concessão de contrato baseada em cálculos de custos marginais determinados por critérios locais de cada agente. O algoritmo é testado numa tarefa de transporte de mercadorias, uma tarefa similar à utilizada nas implementações realizadas nesta dissertação. No algoritmo, um agente submeterá uma oferta para um conjunto de tarefas de transporte somente se o preço máximo mencionado no anúncio da tarefa é maior que o custo no qual incorreria o agente para realizar a tarefa. Uma motivação simples é utilizada para incentivar os agentes a ofertar: o preço real do contrato que é a metade da diferença entre o preço mencionado no anúncio da tarefa e o preço da oferta (que é uma função do custo da tarefa).

Todos os protocolos estudados nesta seção estão baseados de alguma forma no CNP. Todos eles, na ausência de informação das capacidades dos agentes, são forçados a fazer consultas

gerais a todos os outros agentes, o que torna as propostas inviáveis para grupos de grande escala, ou contratar qualquer agente, o que resulta em alocações ineficientes.

Em geral, toda contratação deve determinar claramente as condições da cooperação, no entanto, para que todas as partes numa contratação concordem e consigam definir essas condições é necessária a utilização de mecanismos de negociação. Na próxima seção será descrito um mecanismo de negociação muito utilizado entre agentes com interesses próprios: os leilões. Como se verá, o protocolo proposto nesta dissertação faz uso de leilões na determinação da recompensa pela realização das tarefas.

2.2.3.1 Leilões

Os leilões existem desde a antigüidade, mas ganharam um impulso muito grande com a internet. Sites como eBay, Amazon, Yahoo!Auction, Priceline, Ubid ou FirstAuction estão se tornando cada vez mais populares. Os leilões que antes eram raros na vida cotidiana, onde se pagavam somas astronômicas por obras de arte ou antigüidades, agora se fazem cada vez mais freqüentes com o ciberespaço. Uma das razões pelas quais os leilões estão sendo cada vez mais populares na internet, é que eles possuem mecanismos simples que são facilmente automatizados. No entanto, apesar de sua simplicidade os leilões são uma poderosa ferramenta que agentes autônomos podem utilizar para negociar.

Um leilão consiste de um leiloeiro e um conjunto de compradores potenciais. O leiloeiro deseja vender um bem ou serviço com o maior preço possível enquanto os compradores desejam obter o bem ou serviço ao menor preço [Sandholm (1999)]. No caso da contratação de agentes o leiloeiro deseja alocar as tarefas pagando a menor recompensa possível e outros agentes capacitados desejam realizar a tarefa recebendo a maior recompensa possível.

Características dos Leilões

O protocolo que governa um leilão pode variar em função de quatro critérios: a forma como se realizam os lances, a determinação do ganhador, o conhecimento dos lances de outros participantes e a forma como o valor real do bem é formado.

Os leilões que podem receber somente um lance de cada participante são chamados leilões de uma rodada (*one-shot*). Os leilões onde os agentes podem realizar sucessivos lances são chamados ascendentes se cada agente começa ofertando um valor pequeno e vai incrementado a oferta até atingir um valor máximo. Outro tipo de leilões é o leilão descendente onde o leiloeiro começa oferecendo o bem por um valor grande e vai diminuindo o valor em sucessivas rodadas.

Outro critério que define o tipo de leilões é a determinação do ganhador. Em alguns leilões o ganhador é o agente que realizou a maior oferta, leilões de primeiro preço (*first-price*). Em outras o ganhador é o agente que realizou a maior oferta, no entanto, o agente ganhador paga o valor da segunda maior oferta, eles são chamados leilões de segundo preço (*second-price*).

Se todos os lances são conhecidos ou não pelos participantes é outro fator que determina o tipo de leilão. Leilões de lances fechados são os leilões onde os lances são privados, e leilões de lances abertos são os leilões onde os lances são conhecidos por todos os participantes.

Existe uma última característica que influi no comportamento do leilão: o valor do bem. O valor do bem é o valor real que um participante dá ao bem leiloado. Esse valor depende de muitos fatores. Quando depende somente de fatores privados, como a necessidade que o agente tem de possuir o bem, o valor é chamado privado. Quando o valor é conhecido por todos os participantes, o valor é chamado público. Outro tipo de valor é o valor correlativo. Nesse caso o valor que um agente dá a um bem depende de uma mistura de fatores privados e valorações de outros agentes.

Tipos de Leilão

Existem quatro protocolos de leilão mais comumente utilizados: leilão inglês, leilão holandês, leilão de primeiro preço com lances fechados (*first-price sealed bid*) e leilão Vickrey.

Leilão Inglês

O leilão inglês é talvez o mais comum. É um leilão de primeiro preço, lance aberto e ascendente. O leiloeiro começa sugerindo um preço de reserva pelo bem. Se os agentes não realizam nenhuma oferta, o bem é alocado ao leiloeiro por esse valor. Os agentes, em cada lance, realizam uma oferta maior que a última oferta realizada. As ofertas aumentam até que nenhum agente volta a realizar um lance. Nesse momento o bem é alocado ao agente que realizou a maior oferta.

A *estratégia dominante* num leilão é a estratégia que conduz ao maior ganho seja para os compradores ou para o leiloeiro. No leilão inglês a estratégia dominante para os compradores é simples: ofertar o menor valor possível que seja maior à última oferta até que a oferta atual atinja o valor calculado do bem.

Leilão Holandês

O leilão holandês é um exemplo de leilão de primeiro preço, lance aberto e descendente. O leiloeiro começa ofertando o bem num valor artificialmente grande (maior ao valor privado esperado de qualquer agente). O leiloeiro continua diminuindo o preço da oferta por algum valor pequeno até que algum agente realize uma oferta igual ao preço atual ofertado. O bem é alocado no agente que realizou a oferta.

Leilões de primeiro preço e lance fechado

Os leilões deste tipo são de uma rodada. Nele, os participantes realizam somente um lance e o bem é alocado ao agente que realizou a maior oferta.

Um problema com este tipo de leilão é que ao final, o ganhador pode considerar como dinheiro perdido a diferença entre sua oferta e a segunda melhor oferta. Portanto a estratégia dominante neste tipo de leilão é ofertar menos do que o valor privado procurando obter informação sobre as possíveis ofertas de outros agentes (especular), e assim evitar ofertar um valor muito pequeno.

Leilão Vickrey

O leilão Vickrey [Vickrey (1961)] é um leilão de segundo preço e de lance fechado. Nele somente existe uma rodada e o bem é alocado ao agente que realizou a maior oferta. No entanto o preço pago pelo ganhador é o valor da segunda maior oferta.

A estratégia dominante num leilão Vickrey de valor privado é ofertar a verdadeira valoração. O agente que oferta um valor maior do que a valoração real pode obter o bem mas corre o risco de pagar um valor maior à valoração. Se o agente oferta um valor menor à valoração terá menos oportunidade de ganhar. Ademais, o agente não obterá nenhum benefício de ofertar um valor menor, dado que pagará o preço da segunda melhor oferta.

2.2.3.2 Contratos

Os contratos são formados entre agentes com interesses próprios que precisam cooperar. A formação de um contrato é um processo complicado que requer protocolos que especifiquem a interação entre os agentes. Para que um contrato seja totalmente definido, os agentes devem determinar elementos como: preços, penalidades e a forma como as tarefas devem ser executadas.

Determinação de preços

As interações entre agentes com interesses próprios que precisam cooperar podem variar desde uma simples compra e venda de bens, até complicadas negociações para a contratação de serviços de outro agente como no caso da contratação de tarefas. Tanto na compra e venda de bens como na contratação de tarefas, os preços podem ser fixos ou negociáveis. Quando os preços são fixos, o melhor que um agente pode fazer é procurar o melhor preço para um bem. Na Internet por exemplo, agentes de software chamados *shopbots* [Markopoulos e Kephart (2002); Greenwald e Kephart (1999)] procuram um bem que satisfaça as necessidades do comprador e que tenha o menor preço possível. Existe também outro tipo de agentes chamado *pricebots* [Greenwald e Kephart (1999)] que realizam a tarefa oposta. Eles determinam um preço para um bem em função das condições do mercado. Depois de analisar o mercado determinam um preço que satisfaça as necessidades de um vendedor. Os shopbots e os pricebots estão sendo cada vez mais utilizados na Internet e espera-se que no futuro sejam uma das principais formas de comércio eletrônico.

Quando os preços dos bens ou serviços não são fixos, os agentes podem se envolver em negociações a fim de estabelecer preços que lhes proporcionem o maior benefício. Um dos

principais métodos de negociação de preços são os leilões (Seção 2.2.3.1). Existem agentes de software que são projetados com o objetivo de assistir a usuários que participam em leilões na Internet. Uma proposta interessante é descrita em [Anthony et al. (2001)]. Nela agentes autônomos participam em múltiplos leilões on-line. A participação em múltiplos leilões permite ter um conjunto mais amplo de escolhas para a compra de bens ou serviços e com isso encontrar melhores preços. No trabalho é apresentado um algoritmo que permite a um agente escolher o melhor leilão e a melhor oferta. O algoritmo 2 define o comportamento que segue um agente comprador nessa proposta.

Algoritmo 2 Algoritmo para o agente comprador

1: **while** $(t \leq t_{max})$ and $(ItemNA = true)$ **do**
2: *Construir lista de leilões ativos*
3: *Calcular atual oferta máxima utilizando a estratégia do agente*
4: *Escolher leilões potenciais a partir da lista de leilões ativos*
5: *Escolher o leilão que maximiza a utilidade esperada do agente*
6: *Ofertar no leilão escolhido*
7: **end while**

O agente comprador recebe dois dados do usuário: o valor privado e o tempo máximo (t_{max}) que pode esperar o usuário para obter o bem. O agente tem um tempo limite para conseguir o bem e, portanto, quanto mais perto do tempo limite, mais agressivas serão as ofertas do agente, a fim de incrementar a oportunidade de obtenção do bem. No algoritmo 2 um agente deixará de procurar um bem se atingiu o tempo máximo t_{max} ou o bem já foi obtido ($ItemNA = false$).

Em todo momento, o agente comprador mantém uma lista dos leilões atualmente ativos. Em todo momento também, calcula a máxima oferta atual. A máxima oferta atual é a oferta que o agente esta disposto a pagar nesse momento num leilão. Ela é calculada tendo em conta critérios como: a quantidade de leilões, o tempo restante até t_{max} ou o nível de desespero do agente. O agente segue uma determinada estratégia. Cada estratégia valoriza mais um critério que outro.

O passo seguinte para o agente é determinar o conjunto de leilões onde pode realizar uma oferta. Os leilões onde poderá ofertar são os leilões ativos onde a oferta atual é menor do que a oferta máxima que o agente está disposto a pagar. Finalmente, depois de determinar o conjunto de leilões potenciais, o agente escolhe o leilão que maximiza a utilidade e oferta nele.

Essa proposta é particularmente interessante, pois, no protocolo proposto nesta dissertação, os agentes também se envolvem em leilões múltiplos. Como será visto, os agentes, dentro do protocolo, realizam as operações descritas no algoritmo 2, no entanto, a forma como são realizadas essas operações muda, dado que são utilizados critérios diferentes para as decisões de cada agente.

Outra proposta onde o preço é determinado por meio de ofertas e contra-ofertas é apresentada em [Fischer et al. (1996)]. Esse trabalho apresenta um protocolo de negociação aplicado

a uma tarefa de transporte cooperativo entre companhias. O protocolo é implementado num sistema chamado MARS. A negociação dá-se em dois níveis: uma negociação vertical entre a companhia e os caminhões de transporte e uma negociação horizontal *peer-to-peer* entre as companhias. A negociação vertical serve para determinar custos e o caminhão mais adequado para realizar a tarefa. A negociação horizontal serve para conseguir uma cooperação entre companhias por meio da determinação de um preço baseado nos custos obtidos na negociação vertical. Essa proposta apresenta também uma extensão do CNP para decomposição e alocação de tarefas. Propõe-se também uma negociação inter-agente com o objetivo de otimizar a solução obtida pelo protocolo. Essa negociação permite tornar o protocolo flexível a mudanças no ambiente, dado que os agentes podem renegociar as tarefas quando as condições de execução mudam.

Esse trabalho é interessante por dois motivos principais: ele faz uso de uma tarefa de transporte como a utilizada nas implementações realizadas nesta dissertação. Além disso, o protocolo apresentado nesse trabalho considera uma cooperação entre agentes com interesses tanto comuns quanto próprios, da mesma forma que o protocolo proposto nesta dissertação. Não obstante, a negociação apresentada em MARS é realizada em três etapas: a negociação horizontal, vertical e a negociação inter-agente. Nesta dissertação propõe-se uma negociação conjunta que integra a cooperação entre times, times-agentes e entre agentes num processo mais simples.

Determinação das condições de execução de tarefas

Outro elemento importante dentro de um contrato é a especificação da forma como deve ser realizada uma tarefa. Kraus (1996) estuda métodos que permitem obter o nível adequado de esforço para a execução de uma tarefa. Os agentes determinam um nível de esforço específico e o que se tenta conseguir é que o agente contratante realize a menor supervisão possível, dado que a supervisão também consome recursos. A contratação de agentes é estudada em diferentes situações: quando os agentes possuem informação completa de cada um e do ambiente, quando os agentes não conhecem parte ou toda a informação do ambiente ou quando os agentes possuem informação privada. Também são considerados encontros repetidos entre agentes e a contratação de grupos de agentes.

Existem outros fatores que determinam a forma como as tarefas são executadas. Em [Collins et al. (2002)] propõe-se uma arquitetura para contratação entre agentes onde as tarefas contratadas possuem restrições de tempo e precedência. A arquitetura é implementada num protótipo chamado MAGNET. Um agente cliente em MAGNET solicita ofertas para a execução de um plano consistente de múltiplos passos e tarefas num *requerimento por quotas*. O requerimento deve incluir as restrições de precedência e tempo para cada tarefa. Os agentes vendedores submetem ofertas que incluem o preço pela realização da tarefa. O agente cliente determina e aceita uma combinação de ofertas que maximize sua utilidade. Essa combinação deve garantir as restrições de tempo e precedência. A infra-estrutura de MAGNET previne fraudes, identidades falsas e evita a especulação.

Penalidades

O que acontece quando um dos agentes não cumpre o contrato? Tradicionalmente os contratos entre agentes dentro de ambientes de negociação automatizados somente admitem um compromisso total. O agente contratante compromete-se a pagar uma recompensa determinada e o agente contratado compromete-se a realizar a tarefa nos termos do contrato sem a possibilidade de sair do contrato no futuro. Eventos futuros podem fazer que um contrato não seja mais benéfico para um agente. Em [Sandholm e Lesser (1995a)] propõe-se um protocolo que admite diferentes níveis de compromisso fazendo uso de penalidades. Um agente pode desistir de um contrato unilateralmente somente pagando a penalidade definida. Esse trabalho mostra formalmente que um protocolo deste tipo possui uma pareto-eficiência maior do que um protocolo com compromisso total (*full commitment*). A penalidade dentro dos contratos é um elemento importante a considerar quando agentes com interesses próprios cooperam.

2.3 Considerações finais

Neste capítulo, foram apresentados conceitos relativos à contratação de agentes e aos swarms. A contratação de agentes foi descrita com base em conceitos como: alocação de tarefas, leilões, o *contract net protocol* e contratos.

Cada tema foi abordado fazendo referência a trabalhos relacionados com o PCSMAGE. Muitas das idéias propostas nesses trabalhos contribuíram no projeto do protocolo proposto nesta dissertação. Especificamente, como será visto no próximo capítulo, o protocolo utiliza o conceito de token e limiar [Scerri et al. (2005)] dentro de um algoritmo que permite descobrir agentes capacitados. O protocolo também utiliza uma mistura do leilão inglês e holandês a fim de definir o preço do contrato. Além disso, cada agente dentro do protocolo envolve-se em interações time-agente, time-time e agente-agente como em [Fischer et al. (1996)] e em leilões simultâneos como em [Anthony et al. (2001)]. Ele também realiza, ainda que utilizando procedimentos distintos, operações como: descoberta de leilões ativos, cálculo de máximas ofertas, escolha de leilões com maior utilidade, etc. Em geral, cada um desses elementos é adaptado a fim de que possam ser utilizados no contexto dos swarms de agentes com interesses próprios.

Capítulo 3

Contratação de Agentes em Swarms

Estamos entrando numa época onde sistemas compostos por uma grande quantidade de agentes que representam os interesses de diferentes entidades são cada vez mais freqüentes. Muitos desses sistemas já estão presentes e serão cada vez mais comuns no futuro. Neste tipo de sistema, muitas vezes um grupo de agentes precisa cooperar com outro grupo para poder atingir os objetivos de cada um. Como ambos os grupos possuem diferentes interesses, faz-se necessária alguma forma de negociação a fim de encontrar um meio-termo que beneficie a todos. A negociação é naturalmente complexa, pois agentes que deveriam cooperar e obter assim um ganho maior, podem não cooperar por causa de conflitos na divisão do ganho. Um agente pode achar que está ganhando muito pouco de acordo com o valor que ele atribui a um bem. Neste ambiente, existem diferentes valorizações de bens ou serviços o que muitas vezes provoca conflitos.

A negociação se torna ainda mais difícil quando não se possui uma visão global do conjunto de possibilidades de negócio, isto é, do conjunto de agentes com os quais se pode negociar e as atividades que eles realizam. Em grupos formados por uma grande quantidade de agentes, somente é possível obter uma visão local do grupo, ou seja, somente é possível ter conhecimento de uma quantidade reduzida de agentes. Obter informações de todos os agentes é uma operação extremamente custosa.

Este capítulo apresenta um protocolo que permite uma cooperação e negociação de baixo custo entre grupos formados por uma grande quantidade de agentes que possuem interesses próprios. Além disso, o protocolo permite também a cooperação entre agentes que têm os mesmos objetivos. Trata-se de um protocolo que possibilita a cooperação por meio do estabelecimento de contratos entre agentes utilizando níveis baixos de comunicação. Geralmente, as tecnologias que usam muita comunicação não são aplicáveis em ambientes formados por uma quantidade massiva de indivíduos.

A próxima seção apresenta uma formalização do problema abordado nesta dissertação. A Seção 3.2 apresenta o protocolo proposto e a Seção 3.3 mostra um exemplo de aplicação do protocolo. As etapas do protocolo são detalhadas nas seções 3.4 e 3.5. O comportamento dos candidatos é definido na Seção 3.6. O cálculo de limiares é apresentado na Seção 3.7. Finalmente, a Seção 3.8 discute as decisões de projeto do protocolo.

3.1 Problema da Contratação em Sistemas Multi-agente de Grande Escala

O *Problema da Contratação em Sistemas Multi-agente de Grande Escala* (PCSMAGE) abordado nesta dissertação pode ser definido como: Dado um swarm formado por vários times $L = \{L_1, ..., L_k\}$ cada um com um interesse diferente, onde cada time é formado por um conjunto de agentes $E_k = \{e_{1,k}, ..., e_{n,k}\}$ e está encarregado de um conjunto de tarefas $\Theta_k = \{\theta_{1,k}, ..., \theta_{m,k}\}$, encontrar de forma eficiente, utilizando níveis baixos de comunicação e processamento, um conjunto de contratos para alocar tarefas em agentes $Cont(e_{n,l}, \theta_{m,k})$ (onde o agente $e_{n,l}$ é contratado para realizar a tarefa $\theta_{m,k}$) que maximize no tempo o ganho total obtido por todos os times. Por ganho total entende-se uma função de utilidade que considera os ganhos parciais obtidos por cada agente e_i dentro do conjunto E de agentes no sistema, pela execução de cada tarefa θ_j dentro do conjunto total de tarefas Θ:

$$\mathcal{G} = \sum_{e_i \in E} \sum_{\theta_j \in \Theta} ((Pag(\theta_j) - Cus(e_i, \theta_j)) * Cont(e_i, \theta_j)), \tag{3.1}$$

onde

$$\forall \theta_j \in \Theta, \textstyle\sum_{e_i} Cont(e_i, \theta_j) \leq 1.$$

$Pag(\theta_j)$ é o pagamento recebido do exterior do sistema, $Cus(e_i, \theta_j)$ é o custo incorrido pelo agente e_i e $Cont(e_i, \theta_j) \in \{0, 1\}$ é o contrato do agente e_i para a execução da tarefa θ_j. O contrato $Cont(e_i, \theta_j)$ toma o valor 1 se o agente é contratado e 0 se não é contratado. A condição da equação 3.1 garante que no máximo um agente executa cada tarefa.

Como em todo protocolo de contratação, o método para encontrar esse conjunto de contratos deve ser simples, pareto-eficiente, individualmente racional e distribuído [Wooldridge (2002)]:

1. Simplicidade: O contrato deve ser simples e deve existir um algoritmo simples para calculá-lo.

2. Racionalidade Individual: Um protocolo possui racionalidade individual se a utilização do protocolo for benéfico para os agentes. Um protocolo é benéfico se garante os interesses dos participantes na negociação. Ser justo (todos os agentes possuírem as mesmas oportunidades) é uma das propriedades que permitem a um protocolo ter racionalidade individual.

3. Pareto-Eficiente: Definido o contrato, não devem existir outros contratos que sejam preferidos por ambos agentes. Isso significa que não existem outros contratos onde um agente aumente a sua utilidade sem que algum outro perca.

4. Distribuição: Um protocolo deve ser projetado para assegurar que não se tem um único ponto de falha (Ex. um único árbitro) e idealmente minimizar a comunicação entre agentes.

3.2 Protocolo para Contratação de Agentes em Swarms

O Protocolo para Contratação em Swarms (PCS) proposto neste capítulo fornece uma solução para o PCSMAGE. Para isso tenta encontrar um conjunto de contratos que permitam alocar tarefas em agentes tentando obter o máximo ganho possível para todos os times, em outras palavras, aproximando uma solução que seja pareto–eficiente. Para que os contratos encontrados sejam pareto–eficientes, deve existir uma boa alocação de tarefas em agentes, ou seja, são executadas as tarefas que possuem maior pagamento e utilizados os agentes melhor capacitados. A alocação que o PCS tenta encontrar não somente é eficiente, 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. A justiça na concorrência e nos preços permitem melhorar a racionalidade individual do protocolo. Também, para poder solucionar o PCSMAGE, o PCS deve ser escalável, ou seja, o projeto do PCS tenta evitar diminuições na eficiência com aumentos na quantidade de agentes.

Além desses existem outros problemas no PCSMAGE, por exemplo: definição de níveis de esforço, penalidades, precedência de tarefas, etc., O protocolo proposto nesta dissertação não pretende solucionar todos os aspectos envolvidos no PCSMAGE, mas sim pretende ser um ponto de partida para futuros protocolos que tentem solucionar todos os problemas encontrados neste contexto.

A Figura 3.1 dá uma visão global do processo que segue o protocolo. 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. Um time pode também individualmente formar um swarm se é composto por uma quantidade grande de agentes.

Dentro do PCS, um agente $e_{i,l}$ do time l contrata um outro agente $e_{j,p}$ do time p para a realização da tarefa $\theta_{k,l,c}$ do time l da classe c (tarefas que requerem agentes com características parecidas são consideras dentro da mesma classe) e paga um aluguel $Al(e_{j,p}, \theta_{k,l,c})$ (calculado em uma negociação entre ambos agentes) ao agente $e_{j,p}$ uma vez que $\theta_{k,l,c}$ é executada. $e_{j,p}$ pode pertencer ao mesmo time l (nesse caso $p = l$) ou a outro time. Quando $e_{i,l}$ contrata um agente $e_{j,l}$ do mesmo time, não é pago aluguel ($Al(e_{j,l}, \theta_{k,l,c}) = 0$) 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 $\theta_{m,r,d}$ pela qual receberia um aluguel $Al(e_{j,l}, \theta_{m,r,d})$ maior.

Os elementos de cada contrato são definidos dentro do PCS (a tarefa que será realizada, o agente que realizará a tarefa e o aluguel pago para a realização da tarefa). 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 concorrer obtém os melhores agentes. O protocolo permite também a definição de aluguéis justos, isto é, regidos pela oferta e a demanda, determinados em um processo de negociação entre os agentes envolvidos.

O processo que o PCS segue para formar o conjunto de contratos passa por três etapas:

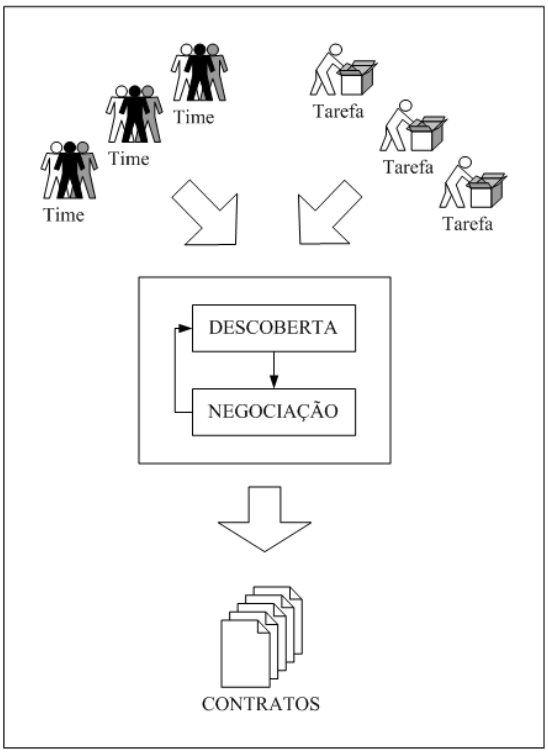

Figura 3.1: Protocolo para contratação de agentes em swarms.

- Descoberta: Cada time l tem em cada momento um conjunto de tokens $\Gamma_l = \{\Delta_{1,l}, \Delta_{2,l}, ..., \Delta_{k,l}\} \forall l \in L$. Cada token contem um possível contrato a ser realizado. Cada vez que uma tarefa $\theta_{k,l}$ é encarregada a um time l, um agente $e_{i,l}$ qualquer do time chamado *administrador* cria um token $\Delta_{k,l}$. O administrador faz percorrer o token pelo conjunto de agentes até encontrar um agente $e_{j,p}$ *candidato* bem capacitado para realizar a tarefa.

- Negociação: O administrador $e_{i,l}$ tenta encontrar um aluguel $Al(e_{j,p}, \theta_{k,l})$ a ser pago a um agente bem capacitado $e_{j,p}$ pela execução da tarefa $\theta_{k,l}$ que maximize o ganho líquido $Gl(\theta_{k,l}) = Pag(\theta_{k,l}) - Al(e_{j,p}, \theta_{k,l})$ do time. O aluguel é calculado por leilões de tarefas e agentes. A maneira como são realizados os leilões será detalhada na Seção 3.5. Desta forma, o preço do aluguel é determinado pelo mercado (oferta e demanda de tarefas e agentes). Se o administrador $e_{i,l}$ e o candidato $e_{j,p}$ não conseguem combinar um aluguel $Al(e_{j,p}, \theta_{k,l})$, $e_{i,l}$ faz percorrer o token novamente em outra etapa de descoberta. Se o candidato $e_{j,p}$ pertence ao mesmo time l do administrador $e_{i,l}$, não existe negociação, $e_{j,p}$ aceita a tarefa se não possui uma outra tarefa com maior pagamento, nesse caso a contratação de agentes seria equivalente a uma alocação de tarefas. Se o candidato $e_{j,p}$ possui outra tarefa com maior ganho, a tarefa de $e_{i,l}$ não é aceita. Nesse momento $e_{i,l}$ deve fazer percorrer o token novamente em outra etapa de descoberta.

- Contrato: Uma vez que ambas as partes (administrador e candidato) estão de acordo no preço do aluguel, o administrador aloca a tarefa, administra a execução da tarefa e

paga o aluguel uma vez que ela é realizada.

A Figura 3.2 mostra a interação dos agentes envolvidos numa execução simples do protocolo. O administrador encarregado de uma tarefa cria um token com o qual percorrerá os agentes. Na etapa de descoberta, o administrador utiliza o token para fazer requisições de informação que permitam determinar a capacidade de cada candidato encontrado. Na etapa de negociação, o administrador oferta aluguéis até o candidato aceitar. Uma vez que o candidato aceita a oferta a tarefa é alocada e o contrato executado.

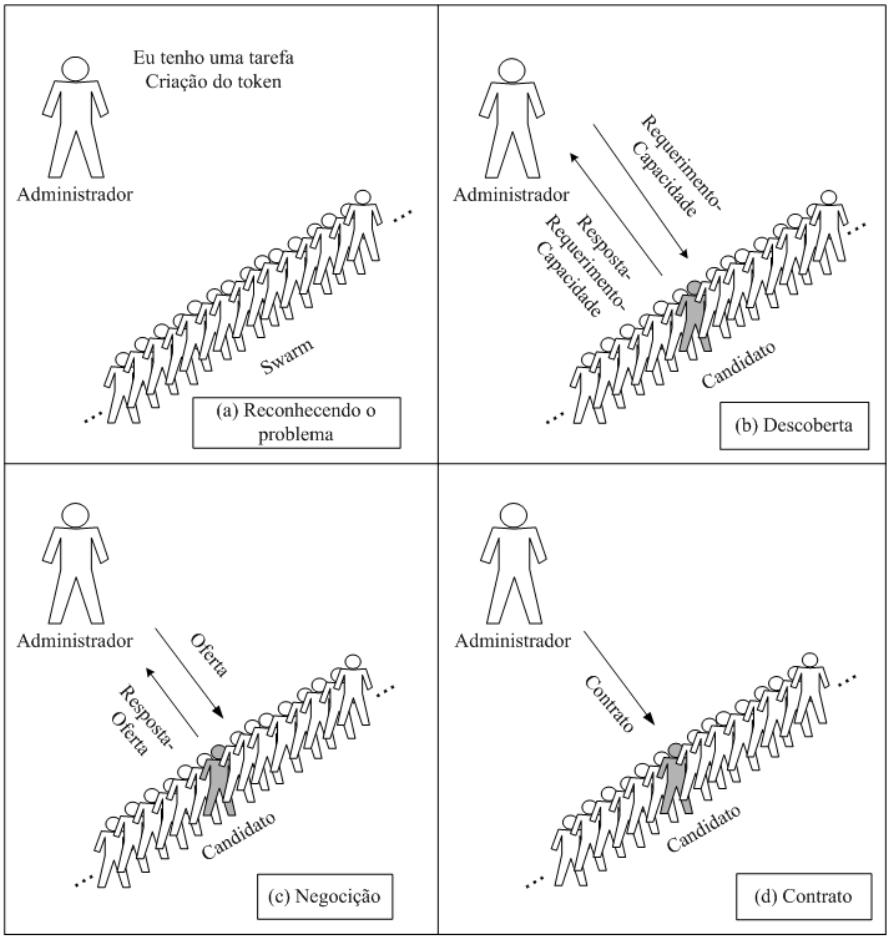

Figura 3.2: Interação no protocolo para contratação em swarms.

A seguinte seção detalha o caso de aplicação que será utilizado de exemplo na descrição das etapas do protocolo nas próximas seções.

3.3 O Problema das Companhias de Transporte de Mercadorias

O problema do transporte de mercadorias [Sandholm e Lesser (1995b); McElroy (1989); Fischer et al. (1996)] consiste na existência de um grande conjunto de veículos autônomos não

tripulados (robôs de transporte) que pertencem a diferentes companhias que se dedicam ao transporte de vários tipos de produtos. Os veículos pegam a mercadoria em uma fábrica determinada e a transferem até um destino. As áreas de operação sobrepõem-se consideravelmente, isso permite que múltiplas companhias possam entregar um mesmo produto.

Dado que múltiplas companhias podem realizar uma mesma tarefa, elas podem intercambiar tarefas a fim de beneficiar-se mutuamente. Uma companhia pode encarregar o transporte de um objeto a outra companhia, se essa última possui veículos que estão mais perto da posição do objeto.

Esta tarefa é um PCSMAGE típico: os agentes são os veículos autônomos, o conjunto de agentes de cada companhia é um time L_k, uma tarefa $\theta_{i,k}$ é cada entrega de um produto x que necessita de um veículo $e_{j,l}$ desde uma origem até um destino e o conjunto total de veículos autônomos é o swarm L.

A tarefa de transporte de mercadorias necessita de uma quantidade grande de agentes, é uma tarefa distribuída e dinâmica dado que tarefas de transporte podem aparecer ou desaparecer continuamente no tempo e é uma tarefa que necessita de negociação dado que as companhias de transporte possuem interesses próprios.

As seguintes seções detalham as etapas do protocolo descrevendo os algoritmos utilizados pelos agentes no processo.

3.4 Descoberta

Quando um time é encarregado de uma tarefa, esta é enviada a um agente administrador. Esse agente é responsável por encontrar um outro agente candidato que esteja capacitado para realizar a tarefa. O administrador pode ser qualquer agente dentro do time. Desta forma, o processo de contratação de agentes é completamente distribuído, característica que contribui para tornar o protocolo escalável. No caso da tarefa de transporte de mercadorias a capacidade que um robô possui pode ser determinada pela distância desde a posição da tarefa até a posição das mercadorias, dado que robôs que estão mais perto terão um custo menor para transportar a mercadoria.

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 todos 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. Seguindo esse método, cada agente administrador no PCS, cria um token e vai percorrendo aleatoriamente com ele um a um os agentes dentro do sistema. Ao chegar a cada um decide, baseado em informação global, se o agente está suficientemente capacitado ou se deve continuar percorrendo os agentes sabendo que encontrará em alguns passos mais um agente com maior capacidade. Na verdade, a infor-

mação global serve para calcular um limiar que discrimina os agentes que estão capacitados para realizar uma tarefa e os que não estão capacitados.

Um token é uma tupla com quatro elementos:

$$\Delta_{i,p} =< \theta_{m,p}, T_{max}, T_{min}, OfertaMax(\theta_{m,p}) >.$$

O token possui dois limiares T_{max} e T_{min} que determinam a capacidade máxima e mínima que deve ter um agente para poder realizar a tarefa. $OfertaMax(\theta_{m,p})$ é a máxima oferta possível em uma negociação (Seção 3.5). Os limiares são calculados no momento da criação do token como será detalhado na Seção 3.7. No PCS, o valor das ofertas (Seção 3.5) bem como a quantidade de agentes no sistema, a quantidade de tarefas e a distribuição de capacidades dos agentes são informações consideradas como públicas, e o custo da realização de uma tarefa, os limiares realmente utilizados, a máxima oferta e o pagamento externo recebido pela realização de uma tarefa são informações mantidas como privadas dentro do time.

O algoritmo 3 detalha o processo que segue um administrador na etapa de descoberta. Ao criar um token $\Delta_{i,p}$ o administrador escolhe 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 $\theta_{i,p,c}$. No PCS a capacidade é uma quantidade normalizada entre 0 e 1 ($Cap(e_{j,l}, \theta_{i,p,c}) \rightarrow [0, 1]$), onde uma capacidade 1 representa a máxima capacidade possível para realizar a tarefa e 0 a mínima capacidade. Se $e_{j,l}$ possui uma capacidade adequada, esse agente é escolhido e considerado como candidato para realizar a tarefa. Um agente $e_{j,l}$ é candidato se $T_{min} < Cap(e_{j,l}, \theta_{i,p,c}) < T_{max}$. Na tarefa de transporte de mercadorias, onde as capacidades estão relacionadas com a distância para o objeto a ser transportado, T_{max} e T_{min} determinarão uma região circular (ver Figura 3.3) ao redor da posição das mercadorias e os robôs administradores procurarão robôs posicionados nessa área.

Algoritmo 3 DESCOBERTA (token Δ)

1: $e_{j,l} \leftarrow EscolheVizinho()$
2: $EnviarMensagem(\text{"}requerimento - capacidade\text{"}, e_{j,l})$
3: **while** $Cap(e_{j,l}, \Delta.\theta_{i,p,c}) < \Delta.T_{min}$ or $Cap(e_{j,l}, \Delta.\theta_{i,p,c}) > T_{max}$ and $\Delta.passos < maxpassos$ **do**
4: **if** $\Delta.passosLimiar > maxpassosLimiar$ **then**
5: $CalcularT_{min}(\Delta)$
6: **end if**
7: $e_{j,l} \leftarrow EscolheVizinho(e_{j,l})$
8: $EnviarMensagem(\text{"}requerimento - capacidade\text{"}, e_{j,l})$
9: **end while**
10: **if** $\Delta.T_{min} < Cap(e_{j,l}, \Delta.\theta_{i,p,c}) < \Delta.T_{max}$ **then**
11: **return** $e_{j,l}$
12: **end if**

No PCS toda solicitação de informação dentro do processo de contratação para uma tarefa determinada, é realizada através de mensagens de requerimento, e cada vez que um agente

envia uma mensagem de requerimento, o agente espera a resposta antes de continuar com o processo de contratação para a tarefa.

O token continua percorrendo agentes (em cada passo o administrador escolhe aleatoriamente um agente vizinho do último agente encontrado) até que um número máximo de passos é atingido ou até que um agente capacitado seja encontrado. Se o número máximo de passos é atingido o token é retirado do sistema. Depois de um período de tempo ou depois que um número de agentes tenha terminado de executar suas tarefas o token pode ser reingressado. O máximo número de passos de um token, o tempo de reingresso e a quantidade de agentes que devem ser liberados para poder reingressar um token são determinados por cada time.

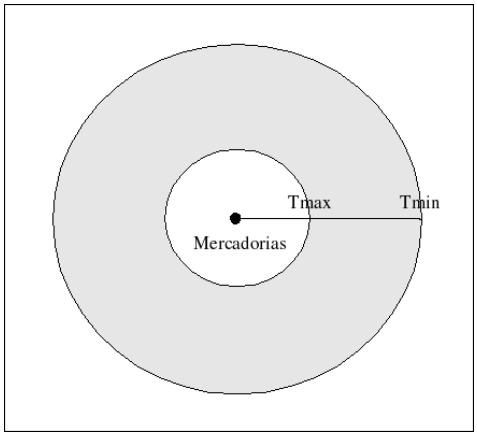

Figura 3.3: Limiares na tarefa de transporte de mercadorias.

O token automaticamente diminui T_{min} depois que atinge um certo número de passos utilizando esse limiar. Se o valor de T_{min} inicialmente calculado estiver inadequado, esta diminuição permitirá novas tentativas para se encontrar um agente adequado. O número de passos antes da correção também é determinado por cada time.

3.5 Negociação

Na negociação os agentes determinam as condições da contratação. No protocolo, especificamente, define-se o preço do aluguel do agente candidato. A negociação está baseada em uma mistura de dois tipos de leilões: leilão inglês e leilão holandês [Sandholm (1999)]. O algoritmo 4 detalha o procedimento que segue um administrador para realizar uma negociação.

O administrador inicia o processo procurando um agente $e_{j,l}$ capacitado (descoberta). 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 $Al(e_{j,l}, \theta_{i,q,c}) = CusME(e_{j,l}, \theta_{i,q,c})$). Por exemplo, na tarefa de transporte, o custo por distância pode depender do valor do combustível utilizado. Nesse caso os administradores utilizarão o valor do combustível mais barato para calcular o aluguel inicial para realizar a primeira oferta aos robôs.

Algoritmo 4 NEGOCIAÇÃO (token Δ, agente $e_{j,l}$)

1: **while** $\Delta.\theta_{i,q,c}$ não alocada **do**
2: $Candidatos = \phi$
3: $CalcularLimiares(\Delta)$
4: **if** $e_{j,l}$ is null **then**
5: $e_{j,l} \leftarrow DESCOBERTA(\Delta)$
6: **end if**
7: $Candidatos \leftarrow Candidatos \cup e_{j,l}$
8: $maxOfertaNivel \leftarrow maxOfertaNivel(Pag(\Delta.\theta_{i,q,c}))$
9: $Al(e_{j,l}, \Delta.\theta_{i,q,c}) \leftarrow AluguelInicial(Cap(e_{j,l}, \Delta.\theta_{i,q,c}))$
10: $EnviarMensagem(\text{"}Oferta\text{"}, Al(e_{j,l}, \Delta.\theta_{i,q,c}), e_{j,l})$
11: **if** $e_{j,l}$ aceita **then**
12: $Contratar(e_{j,l})$
13: **return**
14: **end if**
15: **repeat**
16: $CalcularLimiares(\Delta)$
17: $e_{h,p} \leftarrow DESCOBERTA(\Delta)$
18: $Candidatos \leftarrow Candidatos \cup e_{h,p}$
19: $Al(e_{h,p}, \Delta.\theta_{i,q,c}) \leftarrow Al(e_{j,l}, \Delta.\theta_{i,q,c})$
20: $EnviarMensagem(\text{"}Oferta\text{"}, Al(e_{h,p}, \Delta.\theta_{i,q,c}), e_{h,p})$
21: **if** $e_{h,p}$ aceita **then**
22: $Contratar(e_{h,p})$
23: **return**
24: **end if**
25: **until** not $Cap(e_{j,l}, \Delta.\theta_{i,q,c}) \cong Cap(e_{h,p}, \Delta.\theta_{i,q,c})$
26: $e_{tmp} \leftarrow e_{h,p}$
27: $Candidatos \leftarrow Candidatos - \{e_{h,p}\}$
28: $Al(e_{j,l}, \Delta.\theta_{i,q,c}) \leftarrow Incrementar(Al(e_{j,l}, \Delta.\theta_{i,q,c}))$
29: **while** $Al(e_{j,l}, \Delta.\theta_{i,q,c}) <= maxOfertaNivel$ **do**
30: **for** each $e_{h,p} \in Candidatos$ **do**
31: $Al(e_{h,p}, \Delta.\theta_{i,q,c}) \leftarrow Al(e_{j,l}, \Delta.\theta_{i,q,c})$
32: $EnviarMensagem(\text{"}Oferta\text{"}, Al(e_{h,p}, \Delta.\theta_{i,q,c}), e_{h,p})$
33: **if** $e_{h,p}$ aceita **then**
34: $Contratar(e_{h,p})$
35: **return**
36: **end if**
37: **end for**
38: $Al(e_{j,l}, \Delta.\theta_{i,q,c}) \leftarrow Incrementar(Al(e_{j,l}, \Delta.\theta_{i,q,c}))$
39: **end while**
40: $e_{j,l} \leftarrow e_{tmp}$
41: **end while**

No momento do cálculo do aluguel inicial o administrador também calcula uma oferta máxima $maxOfertaNivel$ para $e_{j,l}$ que é uma função do pagamento que será recebido ($maxOfertaNivel = f(Pag(\theta_{i,q,c}))$). 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 $maxOfertaNivel$. As ofertas são incrementadas com alguma quantidade fixa δ_q definida pelo time. 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 limiares como detalhado na Seção 3.7 para verificar se é possível encontrar ainda um outro agente capacitado e com um custo aceitável. Todo token possui uma oferta máxima $OfertaMax$ que é o maior aluguel que pode ser oferecido a qualquer agente no sistema (a diferença de $maxOfertaNivel$ que é o maior aluguel que pode ser oferecido somente aos agentes dentro do conjunto de $Candidatos$). $OfertaMax$ 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 $Cus(e_{j,l}, \theta_{i,q,c}) <= OfertaMax(\theta_{i,q,c})$. Depois do recalculo de limiares, se não existirem 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 limiares iniciais relativamente altos. Eles oferecerão, para agentes bem capacitados, aluguéis inicialmente baixos, até oferecer, se necessário, quase o pagamento total pela realização da tarefa. Também neste processo todos os agentes candidatos receberão uma oferta igual antes 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 $Al(e_{j,l}, \theta_{k,l,c}) = Cus(e_{j,l}, \theta_{k,l,c})$. Quando $e_{j,l}$ avalia suas ofertas, o aluguel ofertado pelo administrador será interpretado como $Al(e_{j,l}, \theta_{k,l,c}) = Pag(\theta_{k,l,c})$, pois o time l receberá o pagamento total do exterior ao alocar a tarefa num agente próprio. Além disso, esse aluguel não é negociado, ou seja, ele não incrementa e somente é ofertado uma vez. 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. Na tarefa de transporte, por exemplo, as companhias podem utilizar robôs próprios se eles estão mais perto das mercadorias do que outros robôs. Nesse caso será necessário utilizar este processo para conseguir uma alocação simples da tarefa ao robô mais perto dentro dos robôs da companhia.

No PCS, 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. 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 fizeram alguma oferta anteriormente e que ainda não alocaram suas tarefas. Essas duas características do protocolo permitem a formação de um leilão inglês onde os administradores ofertantes tendo conhecimento da melhor oferta até o momento podem oferecer um aluguel maior. A Seção 3.6 detalha o comportamento que possui todo candidato. Nela pode se perceber mais claramente a formação do outro leilão presente no protocolo, o leilão holandês.

3.6 O Papel do Candidato

As seções anteriores descreveram as duas etapas do PCS em função do comportamento do administrador. Esta seção detalha o comportamento dos candidatos dentro do processo e a interação que eles têm com os administradores.

A principal atividade que os candidatos realizam dentro do protocolo é a avaliação das ofertas recebidas. O algoritmo 5 detalha o comportamento dos candidatos frente às ofertas dos administradores.

Um candidato pode recusar uma oferta se: ele já está alocado, o aluguel oferecido é menor do que o aluguel da maior oferta já recebida ou o tempo de espera t_q da oferta é menor que o tempo mínimo tm_l que um agente deseja esperar antes de aceitar qualquer oferta. O tm_l é determinado por cada time l. Os agentes do time não aceitarão ofertas com tempos de espera menores ($t_q < tm_l$). O candidato também verificará se existem outros agentes que possam aceitar a oferta (Ex. Robôs que estão mais perto), ou seja, agentes concorrentes. Se existe pouca concorrência, o agente pode recusar a oferta com uma probabilidade (calculada por meio da função *Concorrencia* no algoritmo 5) inversamente proporcional à quantidade de agentes capacitados que podem aceitar a oferta, dentro do conjunto total de agentes no sistema E:

$$\mathcal{P}(recusar) = 1 - \frac{\mid Concorrentes \mid}{\mid E \mid}, \tag{3.2}$$

onde

$$\mathcal{C}oncorrentes = \{e_i | \Delta.T_{min} < Cap(e_i, \theta_j) < \Delta.T_{max}, CusME(Cap(e_i), \theta_j) <= Al(e_i, \theta_j))\}. \tag{3.3}$$

A quantidade de agentes concorrentes é uma função da distribuição de capacidades dos agentes e do custo das tarefas. Dado que o custo das tarefas é uma informação privada, os agentes candidatos podem supor que o custo incorrido pelos agentes concorrentes é igual ao custo menor esperado numa estratégia adversa ao risco ou é igual ao custo máximo numa estratégia mais riscosa.

Desta maneira os administradores começarão ofertando aluguéis pequenos tentando 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 (neste caso com ofertas crescentes) concorrendo por tarefas.

Toda oferta tem um tempo de espera (t_q). O candidato pode utilizar todo esse tempo antes de aceitar a oferta a fim de esperar melhores ofertas ou pode aceitar a oferta antecipadamente. Toda vez que um candidato obtém uma máxima oferta (uma oferta maior que todas as outras já recebidas), comunica o aluguel dessa oferta aos agentes 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 ($t_{atual} \leq t \leq t_{atual} + t_q$).

Algoritmo 5 AVALIAÇÃO DE OFERTAS (administrador $e_{i,q}$, oferta $Al(e_{j,l}, \Delta.\theta_{i,q,c})$, token Δ)

1: **if** *alocado* **then**
2: recusar oferta
3: **return**
4: **end if**
5: $Ofertantes \leftarrow Ofertantes \cup e_{i,q}$
6: **if** $Al(e_{j,l}, \Delta.\theta_{i,q,c}) <= MaxOfertaRecebida$ **then**
7: recusar oferta
8: **return**
9: **else**
10: **if** $Al(e_{j,l}, \Delta.\theta_{i,q,c}) < Cus(e_{j,l}, \Delta.\theta_{i,q,c})$ **then**
11: recusar oferta
12: **return**
13: **else**
14: **if** NOT $Concorrencia(Al(e_{j,l}, \Delta.\theta_{i,q,c}), \Delta.\theta_{i,q,c}))$ or $t_q < tm_l$ **then**
15: recusar oferta
16: **return**
17: **else**
18: $MaxOfertaRecebida \leftarrow Al(e_{j,l}, \Delta.\theta_{i,q,c})$ {reter oferta, a oferta pode ser aceita ou recusada posteriormente e antes da espera máxima.}
19: **for** each $ofertante \in Ofertantes$ **do**
20: $EnviarMensagem(\text{"}MaxOfertaRecebida\text{"}, ofertante)$
21: **end for**
22: **end if**
23: **end if**
24: **end if**

3.7 Cálculo de Limiares

Esta proposta do PCS considera que cada agente pode executar somente uma tarefa em cada momento. Portanto todo agente pode aceitar somente uma tarefa em cada alocação. Considera-se também que o cálculo de limiares é independente para cada classe de tarefa. Uma classe de tarefa pode ser um quadrante se dividirmos o espaço onde operam as companhias de transporte de mercadorias em quadrantes. Então cada tarefa de transporte que está dentro de um quadrante pertence a essa classe.

Todo token $\Delta_{i,A} =< \theta_{m,A}, T_{max}, T_{min}, OfertaMax(\theta_{m,A}) >$ possui dois limiares T_{max} e T_{min} que representam a capacidade máxima e mínima que deve ter um agente para realizar a tarefa. Os limiares bem como a capacidade são uma quantidade normalizada entre 0 e 1 ($T_{max} \rightarrow [0,1]$,$T_{min} \rightarrow [0,1]$). O cálculo de cada limiar está baseado em dois elementos: a distribuição de capacidades dos agentes e o nível de concorrência. Eles determinam os dois tipos de cálculo de limiares considerados no protocolo: cálculo baseado na distribuição de capacidades e cálculo baseado na concorrência. Além disso, é proposto um recálculo de limiares executado cada vez que se realiza uma descoberta de agentes.

3.7.1 Cálculo baseado na distribuição de capacidades

O cálculo de limiares 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.

Esta proposta do protocolo considera uma distribuição de capacidades uniforme (robôs distribuídos uniformemente na área de execução das companhias de transporte) onde $T_{max} = 1$ e a fórmula para calcular T_{min} é a seguinte:

$$T_{min} = \begin{cases} 0 & \text{if } \mid \Theta_{l,c} \mid > \mid E \mid \\ 1 - \frac{|\Theta_{l,c}|}{|E|} & \text{if } \mid \Theta_{l,c} \mid <= \mid E \mid \end{cases} \tag{3.4}$$

Se a quantidade de tarefas $\mid \Theta_{l,c} \mid$ da classe c do time l é maior do que a quantidade total de agentes $\mid E \mid$ no sistema, T_{min} permite considerar todos os agentes do sistema. Em outro caso, T_{min} permite considerar uma porcentagem de agentes suficiente para executar todas as tarefas da classe.

Este limiar não considera tarefas de outros times e permite que um time possa concorrer por agentes capacitados mesmo não tendo conhecimento de limiares utilizados em execuções prévias do protocolo. Esses elementos são considerados no cálculo de limiares baseado na concorrência detalhado na próxima seção.

3.7.2 Cálculo baseado na concorrência (*Learned Threshold*)

Um cálculo de limiares 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. Como detalhado na Seção 3.7.1, três são os dados básicos necessários para poder calcular o threhosld: quantidade de tarefas concorrentes no sistema, quantidade de agentes e distribuição de capacidades nos agentes.

No caso de agentes com interesses próprios, os times mantém privadas certas informações, especialmente o valor do pagamento recebido $Pag(\theta_{i,l})$. Isso porque no caso em que o pagamento seja maior do que o pagamento médio recebido pela realização de uma tarefa dessa classe, os agentes que finalmente realizam a tarefa, poderiam querer obter parte desse valor, gerando desacordos na divisão do ganho. Além disso, no caso em que o pagamento é menor do que o pagamento médio, os times podem não ter confiança nessa informação. Dado que um time que recebe mais por uma tarefa pode pagar um aluguel maior, o dado da quantidade de tarefas concorrentes (tarefas que podem ganhar agentes que o outro time pode necessitar) é desconhecida.

Para solucionar este problema foi considerado um cálculo de limiares baseado em execuções prévias do protocolo. Este cálculo é chamado *Learned Threshold* (Limiar Aprendido). Esta proposta do PCS considera um cálculo simples do Learned Threshold realizado da seguinte maneira: depois de uma execução satisfatória (com contratação) ou não satisfatória (sem contratação) do protocolo com cálculo de limiares baseado na distribuição de capacidades (ver Seção 3.7.1), o time armazena o maior limiar utilizado T_{max}, assim como o menor limiar utilizado T_{min} para alocar todas as tarefas de cada classe. Depois dessa etapa o time calcula e armazena a quantidade de agentes com capacidade maior do que T_{max} ($\#Maiores$) e a quantidade de agentes com capacidade menor do que T_{max} e maior do que T_{min} ($\#Concorrentes$) para uma determinada distribuição de capacidades. $\#Maiores$ e $\#Concorrentes$ junto com uma distribuição de capacidades são utilizados para o cálculo de limiares em próximas execuções. Esse método permite que os limiares utilizados numa execução sejam recuperados em próximas execuções mesmo existindo mudança no número de agentes.

$$\#Maiores = F(Tmax, Distrib\_de\_Capacidades), \tag{3.5}$$

$$\#Concorrentes = F(Tmin, Distrib\_de\_Capacidades), \tag{3.6}$$

$$\mathcal{T}max = F(\#Maiores, Distrib\_de\_Capacidades), \tag{3.7}$$

$$\mathcal{T}min = F(\#Concorrentes, Distrib\_de\_Capacidades). \tag{3.8}$$

$\#Maiores$ é utilizado para determinar um T_{max} que reserve, para a próxima execução, $\#Maiores$ agentes para as tarefas de outros times (agentes que não puderam ser utilizados na execução atual dado que tinham melhores ofertas de outros times, eles representam os leilões perdidos do time). $\#Concorrentes$ é utilizado para determinar um T_{min} que reserve $\#Concorrentes$ agentes a partir de T_{max}. Os agentes com capacidades entre T_{max} e T_{min} são utilizados tanto para tarefas do próprio time quanto de outros times. No caso da

tarefa de transporte de mercadorias, por exemplo, as companhias armazenam $\#Maiores$ e $\#Concorrentes$ e determinam, dessa maneira, uma região circular ao redor do ponto central no quadrante de cada classe de mercadorias (ver Figura 3.3). Nas próximas alocações os administradores procurarão diretamente robôs nessa região.

Cada vez que um agente cria um token, ele calcula limiares baseados na concorrência se possui informação de limiares utilizados em execuções prévias ou calcula limiares baseados na distribuição de capacidades se não possui informação de limiares prévios. Nesta proposta do protocolo, um time utiliza o limiar baseado na concorrência se a quantidade de tarefas de outros times não mudou acima de um valor máximo. Se a quantidade de tarefas mudou mais do que esse valor, os limiares somente são calculados baseados na distribuição de capacidades. Isso porque o limiar baseado na concorrência ainda não conta com um procedimento para se adaptar a diminuições no número de tarefas de outros times, o seja, a diminuições na concorrência.

O cálculo do learned threshold pode ser melhorado utilizando procedimentos mais sofisticados como os referidos a séries temporais ou com a utilização de outros parâmetros estatísticos (média, desvio padrão) além dos valores máximo e mínimo utilizados. O learned threshold pode servir não somente para encontrar um melhor limiar para a contratação de agentes, mas também para corrigir limiares num processo normal de alocação aproximada de tarefas num time cooperativo, como o caso da alocação em times de larga escala. Nesses times, muitas vezes a informação global que se tem é inexata ou ainda, mesmo possuindo toda a informação, a criação de uma fórmula que contemple todos os casos é complicada, sobretudo uma fórmula que seja adequada para toda distribuição de capacidades. Nesses casos a utilização de informação obtida em processos prévios facilita o cálculo de um bom limiar.

3.7.3 Recálculo de Limiares

Sempre que o administrador realiza uma nova descoberta de agentes os limiares utilizados no token devem ser recalculados a fim de determinar se é possível ainda encontrar agentes capacitados para a tarefa. Um conjunto de novas descobertas são necessárias sempre que se quer formar um novo conjunto de candidatos, uma vez que o conjunto de candidatos anterior recusou as ofertas do administrador.

A fim de procurar o primeiro agente do novo conjunto de candidatos, na proposta atual do protocolo, o limiar máximo é igualado à capacidade do último agente encontrado considerando como busca esgotada a busca num conjunto de agentes com capacidades entre o limiar máximo e a capacidade do último agente encontrado. Também $\#Maiores$ é recalculado com base no novo T_{max} e T_{min} deve permitir reservar $\#Maiores+ \mid Concorrentes \mid$ agentes.

O recálculo do limiar mínimo executado a fim de procurar outros candidatos considera a distribuição de capacidades e a quantidade de agentes candidatos $\mid Candidatos \mid$ a fim de determinar se é possível ainda encontrar agentes capacitados.

3.8 Decisões de Projeto

As próximas seções justificam algumas decisões que foram tomadas para o projeto do protocolo.

3.8.1 Leilões

O protocolo utiliza o leilão inglês e holandês. Leilões de oferta selada como o caso das ofertas Vickrey, possuem a desvantagem de que os ofertantes revelam informação privada ao ofertar diretamente seus valores privados [Rothkopf et al. (1990)]. No caso da contratação de agentes para a execução de tarefas, os agentes administradores deveriam ofertar um aluguel muito próximo ao pagamento recebido pela tarefa. Isso poderia provocar uma situação onde outros agentes, como os que finalmente realizam a tarefa, queiram obter parte do ganho adicional no caso de que o pagamento pela tarefa seja maior que o esperado. O protocolo sempre conserva privada a informação dos times. Não revela informação confidencial como o caso do pagamento obtido ou os custos incorridos por cada agente na realização de cada tarefa. Se o protocolo obrigasse a revelar os custos dos agentes, times eficientes que conseguem reduzir seus custos poderiam estar expostos a agentes que queiram obter também parte deste benefício. Esse fato poderia adicionalmente provocar conflitos na divisão do ganho que dificultariam a cooperação.

Além disso, como o protocolo utiliza informação global, os administradores poderiam utilizar esta informação para especular antes de realizar as ofertas e ofertar assim valores inferiores. Dessa forma, e dado que o controle do fluxo de informação entre agentes num swarm não é uma tarefa trivial, os agentes candidatos poderiam enviar as melhores ofertas recebidas aos administradores ou até criar ofertas fictícias "shills"[1] a fim de aumentar o preço do aluguel. Com isso os administradores poderiam contra-ofertar e dessa forma evitar o uso de leilões de uma só roda (one-shot) como os leilões de oferta selada. Dentro do protocolo os times utilizam a informação global para calcular a probabilidade da existência de cada oferta comunicada, dessa forma podem perceber ofertas fictícias.

3.8.2 Limiares Globais

Uma alternativa para o cálculo do limiar seria levar em conta todas as tarefas dentro do sistema. Um limiar calculado assim alocaria eficientemente as tarefas com baixa comunicação [Scerri et al. (2005)]. No entanto, apesar de que os times ainda poderiam competir em leilões, muitas tarefas que possuem um ganho maior e puderam ser alocadas em melhores agentes poderiam encontrar e ficar alocadas em agentes pouco capacitados. Isso devido ao fato que todos os tokens para todos os times possuiriam o mesmo limiar. O uso deste tipo de limiar tornaria o protocolo pouco justo, uma característica não desejável em protocolos de negociação.

[1] maior informação sobre o tema em [Sandholm (1999)]

3.8.3 Roteamento de tokens

Uma alternativa ao uso de limiares é o uso de roteamento de tokens baseado em modelos históricos da atividade de cada agente no sistema. Um algoritmo que utiliza esse tipo de roteamento foi apresentado em [Xu et al. (2006)]. Cada token é inteligentemente roteado com base em modelos que armazenam a atividade dos vizinhos de cada agente. Em tarefas onde a capacidade dos agentes muda continuamente, não parece adequado usar modelos históricos que se tornariam rapidamente desatualizados. Por exemplo, a tarefa de transporte de mercadorias, onde a capacidade dos agentes é determinada pela distância desde o agente até a mercadoria a ser transportada e onde os agentes mudam de posição cada vez que realizam um transporte, não se beneficia de modelos históricos que guiariam os tokens das tarefas até agentes que se encontram longe, porém que alguma vez se encontraram perto da tarefa.

Além disso, em sistemas onde a quantidade de tarefas aumenta com o tempo, algoritmos de roteamento de tokens conduziriam os tokens das novas tarefas até agentes já ocupados. Todos os tokens seriam dirigidos ao pequeno grupo de agentes utilizados na alocação anterior. Essa é outra desvantagem que tornou pouco atraente o uso desse tipo de algoritmo no protocolo.

3.9 Considerações Finais

Este capítulo apresentou o PCS, um protocolo para a contratação de agentes em swarms proposto nesta dissertação. Foram descritas as etapas e cálculos necessários para a execução do protocolo bem como a interação das partes envolvidas.

Foram apresentadas também as justificativas para algumas decisões de projeto do protocolo a fim de que o leitor possa ter uma idéia mais clara do objetivo de cada algoritmo.

Capítulo 4

Resultados Experimentais

Este capítulo apresenta experimentos realizados com o objetivo de determinar em que medida o PCS resolve o PCSMAGE. Para que o protocolo possa dar solução ao PCSMAGE é necessário que seja escalável e flexível e que, além disso, possua características desejáveis em todo protocolo de negociação: racionalidade individual, simplicidade, pareto-eficiência e distribuição.

Na próxima seção detalha-se as implementações realizadas para os experimentos: implementação do PCS, implementação da tarefa de transporte utilizada nos testes e a implementação, para efeitos de comparação, de um protocolo típico de contratação em $SMAs$, a Contratação Gulosa. Além disso, a seção descreve a ferramenta utilizada para os experimentos, um simulador multi-robô chamado MuRoS. Na Seção 4.2 detalha-se a forma como foram realizados os experimentos e os parâmetros utilizados e na Seção 4.3 analisam-se os resultados obtidos em cada tipo de experimento.

4.1 Implementação

Para a execução dos experimentos, foram realizadas simulações de uma tarefa de transporte de mercadorias comumente utilizada em pesquisas em coordenação e negociação [Sandholm e Lesser (1995b); McElroy (1989); Fischer et al. (1996)] (Ver Seção 3.3). Trata-se de uma tarefa onde robôs móveis transportam objetos encarregados por clientes desde uma posição origem até um destino. Este domínio é interessante porque permite observar o comportamento do protocolo em um grupo de agentes situados, o que define uma tarefa no contexto da robótica.

Na implementação realizada para este problema foram feitas as seguintes suposições: que as tarefas de transporte chegam ao sistema dinamicamente ao longo do tempo, cada robô móvel somente pode realizar uma tarefa de cada vez, e a capacidade dos robôs é dada por uma função da distância desde o robô até a posição das mercadorias, portanto dado que a capacidade dos robôs é determinada pela sua posição, os robôs mudam dinamicamente de capacidade em cada tarefa executada. Nos experimentos, o custo para realizar cada tarefa de transporte foi calculado utilizando um custo por distância (distância de cada robô até a posição das mercadorias) diferente para cada time, as constantes para o incremento do aluguel

e o decremento de limiares foram as mesmas para todos os times e o pagamento externo pelas tarefas dependia do time ao qual pertenciam (cada time possuía um pagamento máximo e mínimo diferente para as tarefas). Nas simulações todos os elementos foram gerados com distribuições aleatórias uniformes a menos que se especifique o contrario. Também não existe preocupação com aspectos físicos das tarefas tais como colisões entre robôs, trajetórias, etc.

4.1.1 MuRoS – Um Simulador Multi–Robô

Geralmente é difícil e caro manter times de múltiplos robôs, especialmente quando se trata de grandes grupos. As simulações em robótica permitem diminuir custos e obter resultados rapidamente antes de se começar as implementações reais. Para esta dissertação foram realizadas simulações numa ferramenta chamada MuRoS (*Multi-Robô Simulator*)[Chaimowicz et al. (2002)]. As implementações do protocolo e da tarefa de transporte de mercadorias no MuRoS permitiram a realização de diversos tipos de testes graças à flexibilidade desta ferramenta e à possibilidade de trabalhar eficientemente com grandes grupos de robôs.

MuRoS é uma ferramenta de software que permite a simulação de times multi-robô executando diversas tarefas tais como manipulação cooperativa, controle de formação, coleta de objetos, etc. Desenvolvido em Visual C++ utilizando programação orientada a objetos, MuRoS permite a implementação de vários tipos de robôs. Novos tipos podem ser facilmente criados herdando características das classes já implementadas. Desenvolvido para o sistema MS-Windows, MuRoS possui uma interface gráfica que permite a construção de diversos ambientes com grande quantidade de elementos entre obstáculos, objetos e robôs. As simulações têm um desempenho muito bom e podem ser observadas em tempo real. Vários parâmetros da simulação podem ser modificados dinamicamente e novos elementos adicionados ao ambiente durante a execução.

4.1.2 Implementação do PCS

A Figura 4.1 mostra a execução do protocolo no MuRoS. Nela pode ser vista uma simulação típica da tarefa de transporte de mercadorias. Na figura, círculos pequenos representam as tarefas de transporte e os círculos maiores os robôs. Tanto as tarefas como os robôs representam o time ao qual pertencem por meio de cores. Os robôs brancos representam os administradores. Robôs que possuem três dados (a tarefa, o time ao qual pertence a máxima oferta e o seu valor naquele momento) são os candidatos. Robôs que possuem um só número (a tarefa contratada), são os robôs que já aceitaram uma oferta. A Figura 4.2 mostra uma situação mais complexa típica dos experimentos com maior quantidade de elementos.

MuRoS simula o paralelismo inerente neste tipo de sistemas por meio da thread que incrementa o tempo de simulação e atualiza o estado de cada elemento. Em cada passo dentro da thread cada robô pode realizar apenas uma operação. Os robôs podem realizar requerimentos de posição a fim de determinar a capacidade de outros robôs, realizar ofertas, responder requerimentos de posição, responder ofertas, verificar o tempo de vencimento (espera máxima) de ofertas recebidas ou enviadas a fim de responder (candidato) ou eliminar (administrador)

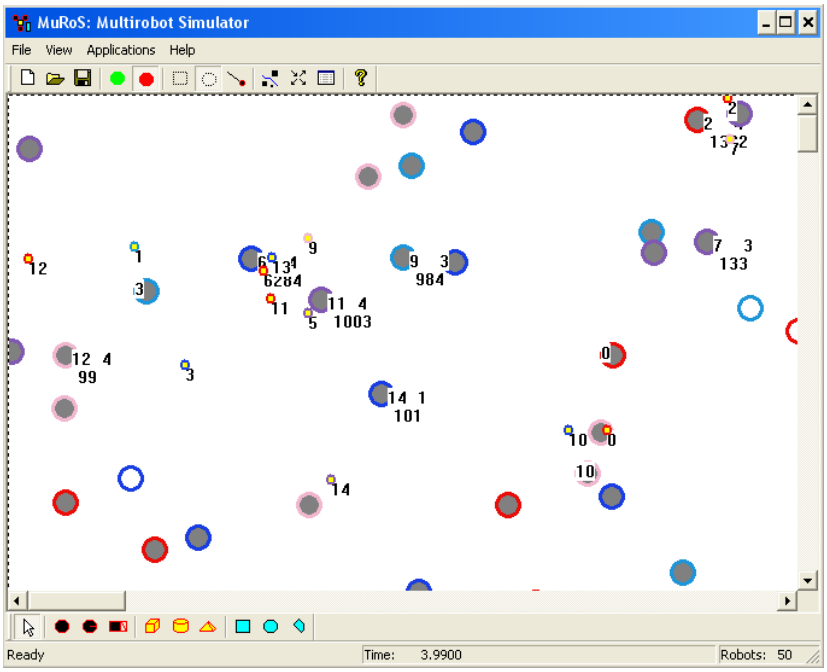

Figura 4.1: Exemplo da Execução do PCS no MuRoS – 50 robôs.

ofertas, verificar respostas de requerimentos de posição e verificar respostas a ofertas. A Figura 4.3 mostra as possíveis interações em uma seqüência simples de mensagens entre robôs na implementação do PCS. Quando um robô realiza um requerimento de posição ou uma oferta não continuará movendo o token que produziu a mensagem, mas pode continuar realizando operações para outros tokens que esteja administrando. Cada vez que um robô realiza um requerimento de posição, uma oferta ou recebe uma oferta que possui possibilidades de ser aceita, o robô cria um processo a fim de verificar a resposta à mensagem e dependendo do estado do robô, ele processará a resposta num momento determinado.

Nesta implementação T_{min} foi diminuído baseado em um número mínimo de passos como descrito no protocolo, e não foram realizados outros tipos de recálculos para esse limiar com exceção do recálculo realizado a fim de procurar o primeiro agente de um novo conjunto de candidatos. Além disso, na implementação do cálculo de limiares baseado na distribuição de capacidades, foi usada uma tolerância ε. Ela permitiu obter um limiar um pouco menos exigente o que resulta na diminuição da quantidade de mensagens utilizadas sem comprometer o ganho total. O limiar mínimo com a tolerância é definido como:

$$T_{min} = \begin{cases} 0 & \text{if } \mid \Theta_{l,c} \mid > \mid E \mid \\ 1 - \frac{|\Theta_{l,c}|}{|E|} - \gamma & \text{if } \mid \Theta_{l,c} \mid <= \mid E \mid \end{cases}, \tag{4.1}$$

onde

$$\gamma = \varepsilon(1 - \frac{\mid \Theta_{l,c} \mid}{\mid E \mid}). \tag{4.2}$$

ε é a tolerância máxima utilizada e γ é a tolerância aplicada que decresce em função da

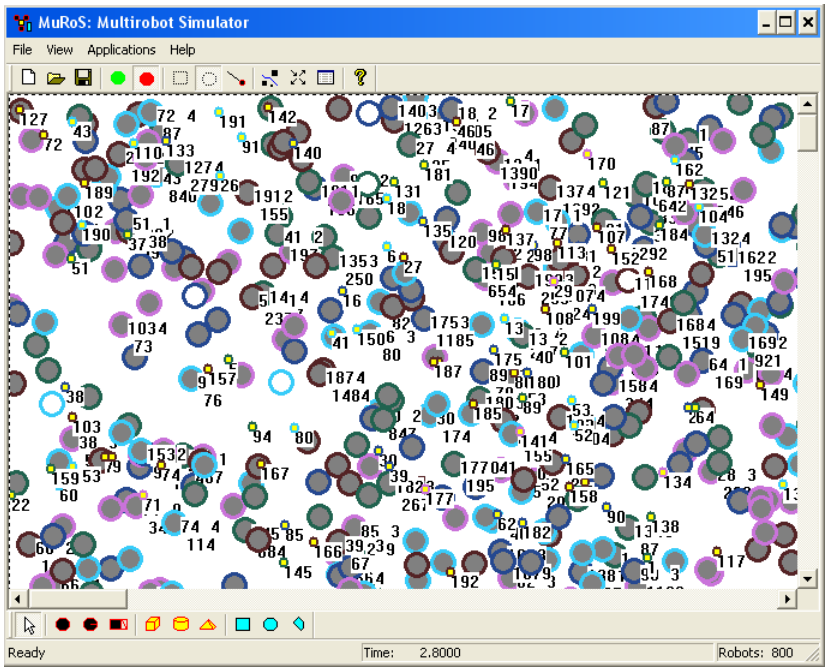

Figura 4.2: Exemplo da Execução do PCS no MuRoS – 800 robôs.

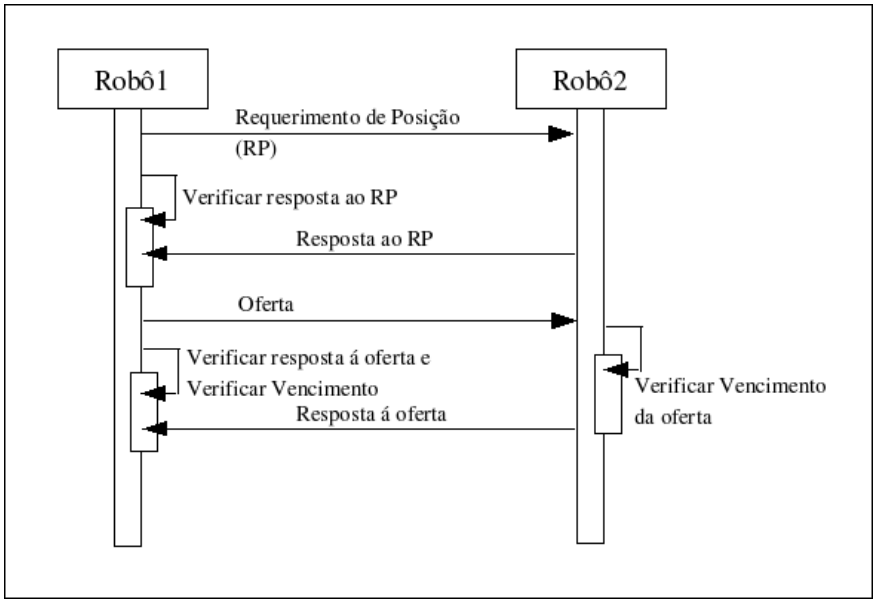

Figura 4.3: Interação entre Robôs na Tarefa de Transporte de Mercadorias.

proporção entre o número de tarefas $\mid \Theta_{l,c} \mid$ da classe c do time l e o número de agentes no sistema $\mid E \mid$.

O restante dos cálculos e algoritmos necessários para a execução do PCS foram implementados como discutidos no Capítulo 3

4.1.3 Implementação da Contratação Gulosa de Agentes

Uma abordagem clássica para a contratação de agentes num SMA é a utilização de um algoritmo guloso que permite encontrar um conjunto de contratos com uma alocação de tarefas próxima à ótima. Uma alocação ótima é uma alocação que permite utilizar o agente melhor capacitado e disponível para realizar cada tarefa e executar as tarefas que possuem o maior pagamento.

Apesar de ser uma implementação gulosa, esse abordagem tende a obter bons resultados dado que utiliza informação de todos os agentes no swarm. O PCS foi comparado com a contratação gulosa (CG) nos testes realizados.

O algoritmo 6 mostra a etapa de descoberta para a abordagem gulosa utilizado nos testes. Na primeira descoberta, o administrador pede informação a todos os agentes no sistema, informação que lhe permite calcular a capacidade de cada agente para realizar a tarefa. Cada vez que o administrador precise encontrar um novo agente para ser incluído dentro das negociações, o algoritmo obtém o próximo agente melhor capacitado entre todos os agentes no sistema. O algoritmo não obterá um agente se já não existirem agentes com um custo aceitável para a realização da tarefa. O algoritmo de negociação é o mesmo utilizado no PCS, mas sem o cálculo de limiares prévio a toda descoberta. O administrador neste algoritmo obtém o melhor agente capacitado e disponível nesse momento para a execução de cada tarefa. Para a contratação de uma tarefa, não se consideram outras tarefas do time ou de outros times atualmente presentes no sistema.

4.2 Metodologia dos Experimentos

Os experimentos realizados têm como objetivo determinar em que medida o PCS resolve o PCSMAGE. Uma característica essencial para se resolver o PCSMAGE é a escalabilidade. Além disso, como descrito na Seção 1.1, existem outras características desejáveis em todo protocolo de negociação. Os experimentos avaliaram também algumas delas, especificamente: racionalidade individual, adaptabilidade e pareto-eficiência. Por fim, algumas características importantes como simplicidade e distribuição foram analisadas de forma não quantitativa.

Algoritmo 6 DESCOBERTA GULOSA (token Δ)

1: **if** $primeira - descoberta$ **then**
2: $atual \leftarrow 1$
3: $R \leftarrow E$
4: **for** each $e_j \in R$ **do**
5: $EnviarMensagem(\text{"}requerimento - capacidade\text{"}, e_j)$
6: **end for**
7: **end if**
8: **if** $Cus(e_{atual}, \Delta.\theta_{i,q,c}) <= OfertaMax(\Delta.\theta_{i,q,c})$ **then**
9: **return** e_{atual}
10: **end if**

Os testes de escalabilidade incluem experimentos para determinar o ganho em escalabilidade produzido pelo cálculo de limiares baseado numa distribuição de capacidades e o ganho produzido pelo cálculo baseado na concorrência. Além disso, foram realizados experimentos que testam a facilidade com que o protocolo adapta-se a mudanças no número de agentes e/ou tarefas. A racionalidade individual foi medida com experimentos que determinam se o protocolo é justo e se rege pela oferta e a demanda. Um protocolo não justo ou onde a oferta/demanda não determinam um preço aceitável não será racional para os times.

As Tabelas 4.1 e 4.2 resumem os valores dos parâmetros utilizados em cada experimento. Esses valores foram determinados de forma empírica. ε é a tolerância utilizada no cálculo de limiares. Custo máximo e mínimo são respectivamente o maior e menor custo por distância que pode incorrer um agente para realizar uma tarefa. Pagamento máximo e mínimo são o maior e menor pagamento que pode obter um time pela execução de uma tarefa de transporte. Espera máxima e mínima são a maior e menor quantidade de passos que um time espera pela resposta a uma oferta. Cada time terá uma espera t_l determinada entre esses dois valores. Na implementação não se considerou uma espera mínima para os times, cada agente responde uma oferta quando ela está por caducar. δ é o incremento do aluguel para os times. Além disso, a área de operação das companhias está dividida por meio de um grid em quadrantes chamados de clusters, cada cluster agrupa fisicamente um número determinado de tarefas. Ele representa a posição física de uma classe de tarefa no sistema. Clusters é a quantidade de quadrantes ocupados do grid onde as tarefas foram agrupadas nos experimentos. Tamanho do cluster é a dimensão física, expressa em unidades quadradas, de uma classe de tarefa. A penúltima métrica, decremento limiar, é o valor que diminui T_{min} cada vez que atinge um máximo número de passos, determinado pela última métrica, o máximo número de passos que um token pode percorrer com um mesmo limiar.

Parâmetro/Experimento	*Escalabilidade*	*L–Threshold*	*Adaptabilidade*
ε	0,05	0,05	0,05
Número de Times	5	7	5
Máximo Custo ($/metro)	1	1	1
Mínimo Custo ($/metro)	0,5	0,5	0,5
Máximo Pagamento ($)	1750	2500	1750
Mínimo Pagamento ($)	250	250	250
Máxima Espera (passos)	250	250	250
Mínima Espera (passos)	240	240	240
δ ($)	50	50	30
Clusters	10–50	3	3
Tamanho Cluster ($metros^2$)	50	50	50
Decremento Lim.	0,03	0.03	0,03
Max passos Lim. (passos)	45	45	45

Tabela 4.1: Parâmetros dos experimentos (Escalabilidade, L–Th e Adaptabilidade).

Parâmetro/Experimento	*Pareto-Eficiência*	*Conc.Justa*	*OfertaDemanda*
ε	0,05	0,05	0,05
Número de Times	5	7	7
Máximo Custo ($/metro)	1	1	1
Mínimo Custo ($/metro)	0,5	0,5	0,5
Máximo Pagamento ($)	1750	2500	1750
Mínimo Pagamento ($)	250	250	250
Máxima Espera (passos)	250	250	250
Mínima Espera (passos)	240	240	240
δ ($)	50	50	30
Clusters	50	3	20
Tamanho Cluster ($metros^2$)	50	50	50
Decremento Lim.	0,03	0.03	0,03
Max passos Lim. (passos)	45	45	45

Tabela 4.2: Parâmetros dos experimentos (Pareto-Eficiência, Concorrência Justa e OfertaDemanda).

4.3 Resultados

As próximas seções descrevem os resultados da avaliação e os experimentos realizados com a implementação do PCS para a tarefa de transporte no simulador MuRoS.

4.3.1 Escalabilidade

Uma das principais métricas que nos permitem determinar se um protocolo é escalável é a quantidade de comunicação utilizada [Xu et al. (2005b)]. Os testes de escalabilidade medem o desempenho do PCS em diferentes configurações tomando em conta tanto a quantidade de mensagens transmitidas pelos robôs quanto o ganho total obtido.

4.3.1.1 Limiar baseado na Distribuição de Capacidades

Esta seção descreve os resultados em escalabilidade obtidos com a utilização de um limiar determinado pela quantidade atual de robôs e tarefas no sistema como explicado na Seção 4.1.2. Os experimentos comparam o desempenho do PCS e a contratação gulosa. Cada teste utilizou uma quantidade diferente de robôs, desde 100 até 5000 robôs, configurações que constituem sistemas multi-agente de grande escala.

As Figuras 4.4 e 4.5 mostram a quantidade de mensagens utilizadas para contratar robôs para realizar 50 e 150 tarefas respectivamente. Resultados similares foram obtidos utilizando quantidades diferentes de tarefas. Para os testes com 50 tarefas foram utilizados 10 clusters e 50 clusters nos testes com 150 tarefas. No eixo x está numerada a quantidade de robôs envolvidos no experimento e no eixo y a quantidade de mensagens utilizadas. Cada ponto na figura é a média de 10 execuções. Ambos algoritmos utilizaram os mesmos conjuntos de robôs e tarefas em cada execução.

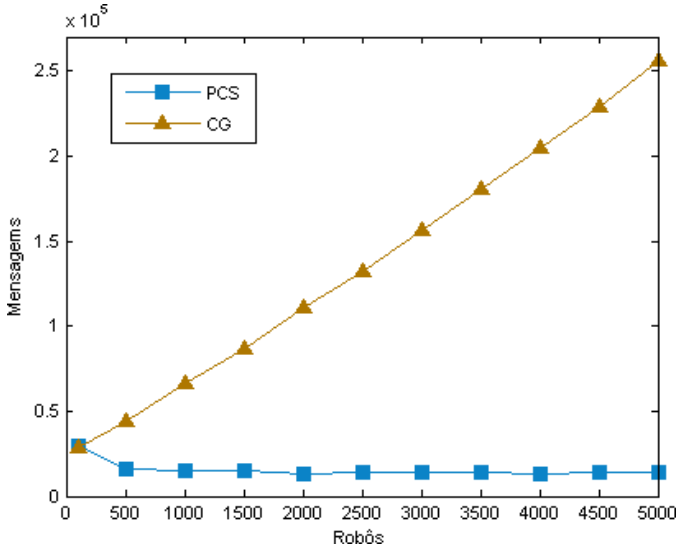

Figura 4.4: Número de mensagens para 50 tarefas.

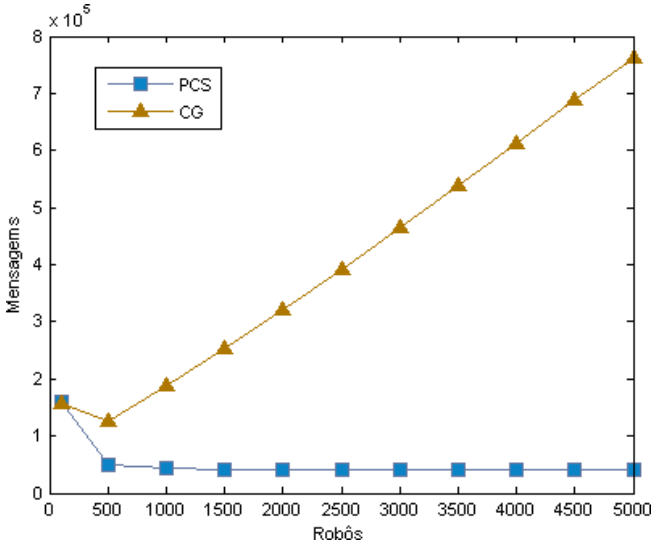

Figura 4.5: Número de mensagens para 150 tarefas.

Os dois gráficos mostram uma quantidade de mensagens muito inferior para o PCS. A quantidade de mensagens no PCS mantém-se constante ao se aumentar a quantidade de robôs no sistema. A quantidade de mensagens na contratação gulosa aumenta linearmente em função da quantidade de robôs. Observa-se também uma diminuição inicial na quantidade de mensagens maior no PCS que na CG. Nessas primeiras configurações a quantidade de mensagens utilizada é maior quando a quantidade de robôs é menor. O motivo dessa diminuição inicial na quantidade de mensagens é que, quando a quantidade de robôs é pequena em relação à quantidade de tarefas, os tokens percorrem o conjunto de robôs por mais tempo antes de encontrar um robô com o qual possam negociar pois, nessas situações, a contratação se torna mais difícil. Quando a quantidade de robôs aumenta a contratação simplifica-se utilizando menos mensagens.

As Figuras 4.6 e 4.7 mostram o ganho obtido nos testes das Figuras 4.4 e 4.5 e o ganho máximo (soma de pagamentos das tarefas) que pode ser obtido. Como esperado, o ganho é maior para a contratação gulosa, mas o PCS mantém também um ganho próximo do ganho máximo.

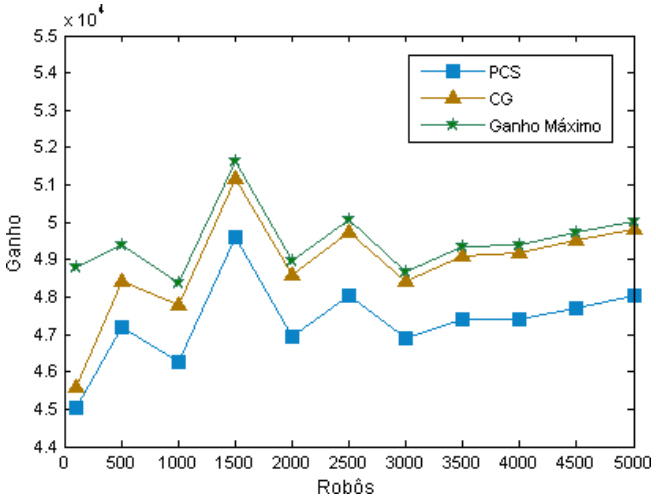

Figura 4.6: Ganho obtido com 50 tarefas.

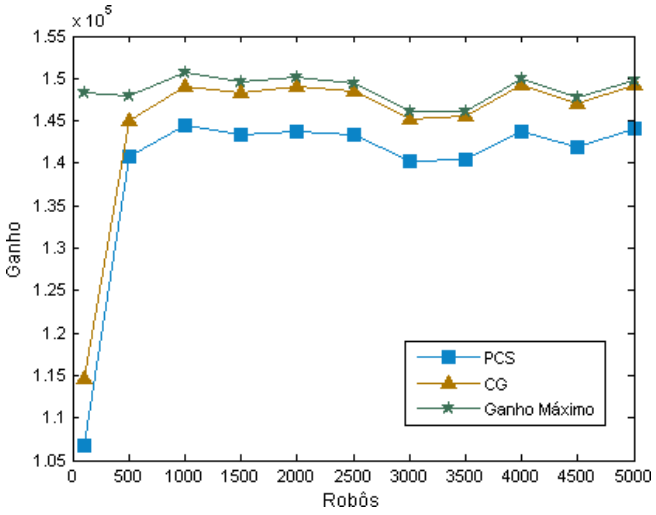

Figura 4.7: Ganho obtido com 150 tarefas.

Nota-se que, em geral, o ganho obtido pelo PCS não é afetado ao se aumentar o número de robôs, com exceção das primeiras configurações onde ambos algoritmos apresentam um ganho inferior do resto das configurações. Esse ganho inferior é provocado por uma quantidade de tarefas que não são executadas devido às dificuldades para se encontrar robôs capacitados quando o número de tarefas é grande em relação ao número de robôs.

Esse primeiro conjunto de experimentos permite observar que o PCS possui uma boa escalabilidade ao manter constante o número de mensagens utilizadas enquanto o número de robôs aumenta e ao manter um ganho próximo ao obtido com uma contratação gulosa.

4.3.1.2 Limiar baseado na concorrência ou Learned Threshold

Esta seção descreve os experimentos de escalabilidade realizados utilizando-se o mecanismo de correção de limiares detalhado na Seção 3.7.2. Eles comparam o desempenho do PCS utilizando limiares baseados numa distribuição de capacidades e o desempenho do PCS utilizando limiares baseados na concorrência (*Learned-Threshold*). Cada teste utilizou entre 100 e 1100 robôs.

As Figuras 4.8 e 4.9 mostram a quantidade de mensagens utilizadas para contratar robôs para realizar 80 e 160 tarefas. No eixo x está numerada a quantidade de robôs envolvidos no experimento e no eixo y a quantidade de mensagens utilizadas. Cada ponto na figura é a média de 5 execuções. Cada execução utilizou dois conjuntos de tarefas. No caso da Figura 4.8 primeiro foi contratado um conjunto de 80 e depois outro lote de 80 tarefas. Do mesmo modo, para os experimentos da Figura 4.9 foram utilizados dois lotes de 160 tarefas. No caso dos experimentos com um limiar baseado na concorrência, o segundo conjunto de tarefas foi contratado utilizando um limiar aprendido na contratação do primeiro conjunto. A quantidade de mensagens em todos os experimentos foi contabilizada somente na contratação do segundo conjunto. Ambos os algoritmos utilizaram os mesmos robôs e tarefas em cada execução.

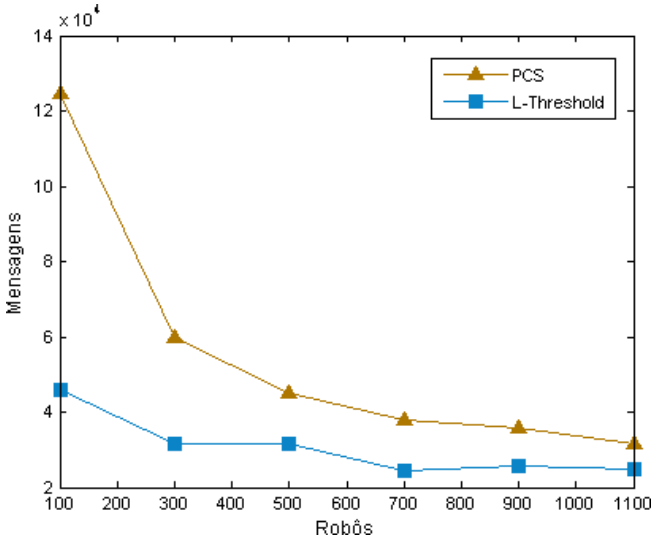

Figura 4.8: Número de mensagens para 80 tarefas.

As duas figuras mostram uma quantidade de mensagens inferior para o learned threshold. Com o learned threshold se consegue diminuir a quantidade de mensagens utilizadas no PCS pois cada time tende a utilizar um intervalo de thresholds ($T_{min}...T_{max}$) diferente, baseado nos limiares utilizados em contratações prévias, isso diminui a coincidência de tokens e, portanto, a realização de leilões. Outro motivo pelo qual a quantidade de mensagens diminui é que o learned threshold corrige a imprecisão própria de um limiar baseado numa distribuição de capacidades, seja no caso de muitos times concorrendo ou apenas um time.

Outra observação importante nas figuras é que a diferença no número de mensagens, entre

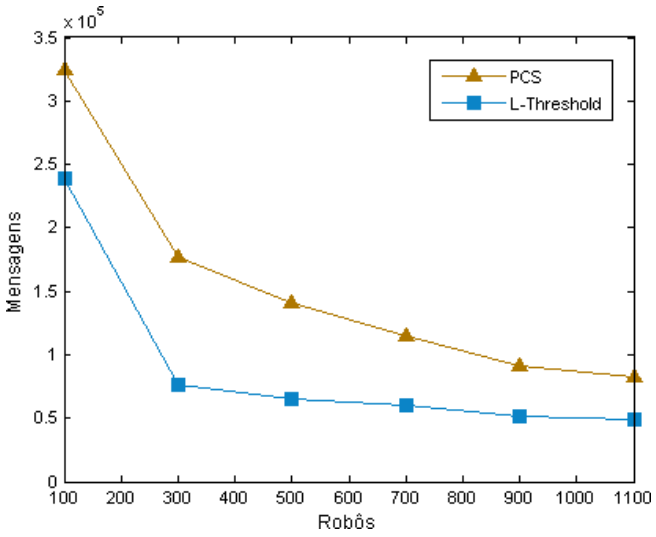

Figura 4.9: Número de mensagens para 160 tarefas.

o PCS com e sem a utilização de um limiar baseado na concorrência, diminui quando a quantidade de robôs aumenta. O aumento na quantidade de robôs para uma mesma quantidade de tarefas provoca uma diminuição da concorrência, fazendo que o erro de limiares calculados sem considerar concorrência seja cada vez menor.

As Figuras 4.10 e 4.11 mostram o ganho obtido nos testes das Figuras 4.8 e 4.9. Os ganhos obtidos com ambos métodos são similares, no entanto, o ganho obtido com learned threshold é um pouco inferior. A execução do protocolo sem learned threshold começa procurando robôs com uma capacidade entre $T1_{min}$ e $T1_{max}$ e contratará robôs com capacidades entre $T2_{min}$ e $T2_{max}$. $T2_{min}$ pode ser aprendido numa contratação com learned thresholds, nessa contratação as tarefas podem ser alocadas em agentes com capacidades iguais, maiores ou mesmo menores do que $T2_{min}$, uma vez que o limiar $T2_{min}$ pode ser diminuido durante o processo de descoberta. As alocações em agentes que possuem capacidades menores do que $T2_{min}$ (que é o menor limiar utilizado nas contratações com limiares baseados na distribuição de capacidades) provocam a diminuição do ganho com a utilização do learned thresholds. É possível diminuir este efeito com a utilização de uma tolerância como se verá na Seção 4.3.2.

Os resultados obtidos neste conjunto de experimentos permitem verificar que o uso de um limiar aprendido é útil para corrigir o erro de um limiar baseado somente numa distribuição de capacidades quando existe concorrência. Isso provoca uma diminuição considerável no número de mensagens sem causar um grande prejuízo no ganho total obtido.

4.3.2 Adaptabilidade

Os ambientes onde os swarms trabalham são extremamente dinâmicos. Eles são ambientes onde as quantidades de agentes e tarefas, as características dos agentes e as condições de execução das tarefas mudam continuamente. Um protocolo para swarms deve ser suficientemente flexível para poder se adaptar a essas condições. Esta seção analisa a adaptabilidade do PCS em ambientes dinâmicos.

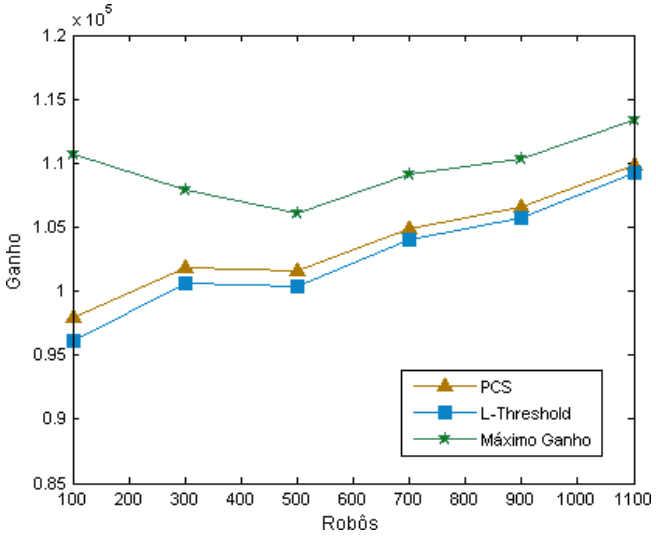

Figura 4.10: Ganho obtido com 80 tarefas.

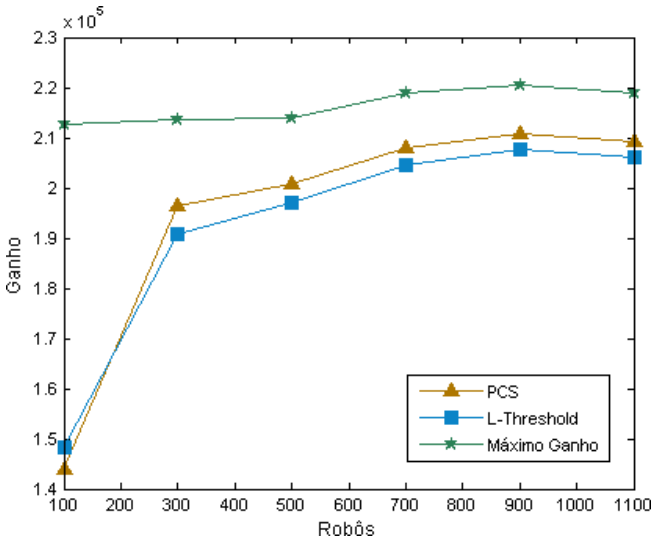

Figura 4.11: Ganho obtido com 160 tarefas.

A versão atual do PCS pode adaptar o learned threshold quando a quantidade de agentes muda, ou quando a quantidade de tarefas aumenta, mas não está preparado para se adaptar a diminuições na quantidade de tarefas. Um possível método para conseguir essa adaptação poderia consistir em percorrer cada token uma determinada quantidade de passos com um limiar sem considerar concorrência. O número de passos poderia ser determinado em função da mudança no número de tarefas. Dessa forma cada token teria a oportunidade de encontrar um agente mais capacitado quando a quantidade de tarefas diminui.

Para tornar o protocolo adaptável, nesta implementação, o PCS utiliza somente um limiar baseado em distribuição de capacidades quando a quantidade de tarefas muda, seja que aumente ou diminua. Em outro tipo de mudanças no ambiente (número de agentes, distribuição de capacidades), o PCS utiliza o limiar baseado na concorrência.

A Figura 4.12 mostra a mudança na capacidade média utilizada por cada time durante uma execução típica do PCS. Por capacidade média entende-se a média das capacidades dos agentes alocados às tarefas do time. Cada time possui um pagamento por tarefa diferente e na figura, os times com maior índice possuem um pagamento médio maior para suas tarefas. Durante a execução a quantidade de tarefas foi diminuída linearmente no tempo medido em segundos simulados no MuRoS. A freqüência com que novas tarefas apareciam era menor do que a freqüência com que as tarefas eram executadas. Existiram também, nesta execução, períodos de tempo onde a quantidade de tarefas não mudou, nesses períodos o protocolo utilizou um limiar aprendido. Na Tabela 4.3 mostra-se a quantidade de tarefas presentes no sistema em cada momento do experimento.

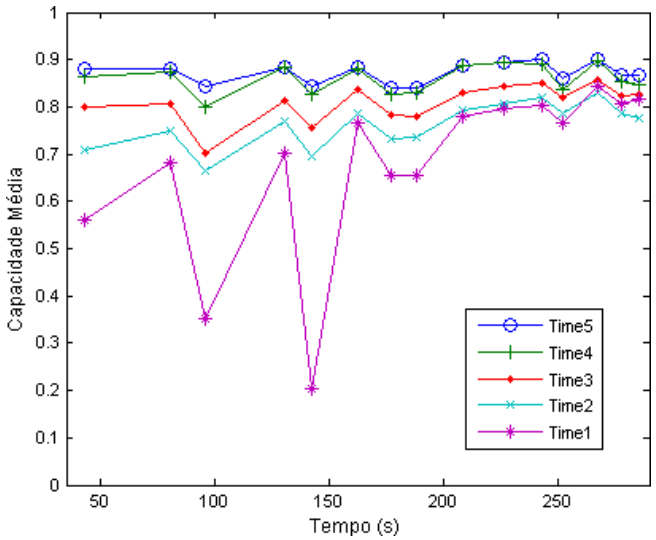

Figura 4.12: Capacidade média utilizada no tempo.

Os resultados mostrados na Figura 4.12 permitem distinguir o aumento da capacidade média utilizada enquanto ocorre uma diminuição na quantidade de tarefas presentes no sistema. A capacidade média aumenta mais em times com menor ganho médio. Em períodos onde a quantidade de tarefas não mudou os times utilizaram uma capacidade média inferior ao limiar aprendido (mostrado pelos pontos inferiores na Figura 4.12), isso também é maior em times com ganho médio inferior.

Na Figura 4.13 mostra-se a mudança na capacidade média considerando somente os períodos de mudança no número de tarefas, períodos onde somente foi utilizado um limiar baseado na distribuição de capacidades. Na Figura 4.13 pode-se distinguir mais claramente o aumento da capacidade média utilizada. Na Figura 4.14 mostram-se somente os períodos onde a quantidade de tarefas permaneceu constante, nela pode-se distinguir o aumento na capacidade média também nesses períodos.

Quando o PCS utiliza um learned threshold, as tarefas são contratadas em robôs com capacidades médias um pouco inferiores às capacidades utilizadas depois de um cálculo de limiares baseado somente na distribuição de capacidades (pontos inferiores na Figura 4.12) como foi explicado na Seção 4.3.1.2. Esse comportamento provoca uma diminuição no ganho

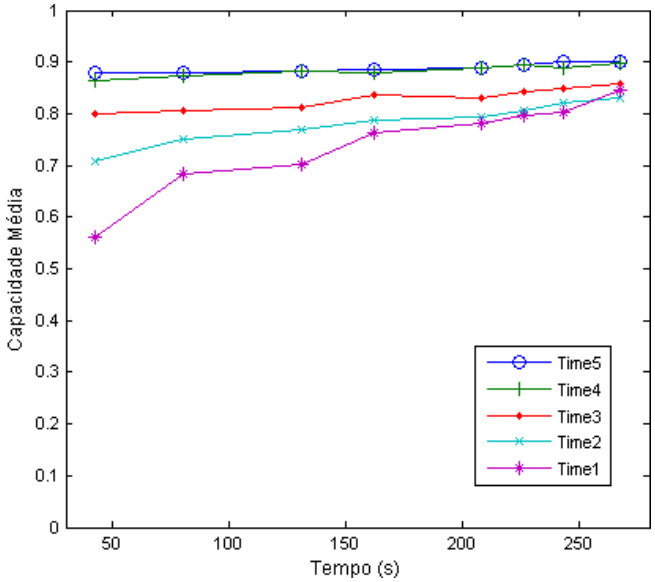

Figura 4.13: Capacidade média no tempo utilizando o cálculo baseado na distribuição de capacidades.

Tempo (s)	*Time1*	*Time2*	*Time3*	*Time4*	*Time5*	*Total Tarefas*
42.65	65	173	142	154	226	760
80.41	56	164	133	145	217	715
95.99	56	164	133	145	217	715
130.86	47	155	124	136	208	670
142.3	47	155	124	136	208	670
162.25	38	146	115	127	199	625
176.96	38	146	115	127	199	625
188.26	38	146	115	127	199	625
208.28	29	137	106	118	190	580
226.15	20	128	97	109	181	535
243.3	11	119	88	100	172	490
252.28	11	119	88	100	172	490
267.29	2	110	79	91	163	445
277.63	2	110	79	91	163	445
285.6	2	110	79	91	163	445

Tabela 4.3: Número de Tarefas no tempo.

obtido (Figuras 4.10 e 4.11). Para solucionar esse problema foi utilizada uma tolerância no learned threshold. Depois de que T_{max} e T_{min} são corrigidos com um cálculo baseado na concorrência, ambos limiares são incrementados com a tolerância. Dessa forma os tokens percorrem o conjunto de robôs com limiares iniciais um pouco maiores que os aprendidos, o que permite que algumas tarefas sejam contratadas em robôs com capacidades maiores aos limiares aprendidos, outras com capacidades dentro do intervalo de limiares aprendidos e outras com capacidades menores aos limiares aprendidos. Em média os tokens utilizarão capacidades dentro do intervalo de limiares aprendidos.

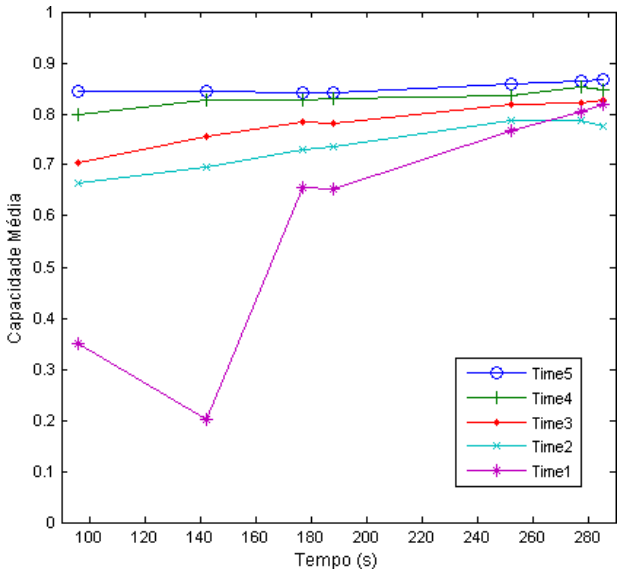

Figura 4.14: Capacidade média no tempo utilizando learned threshold.

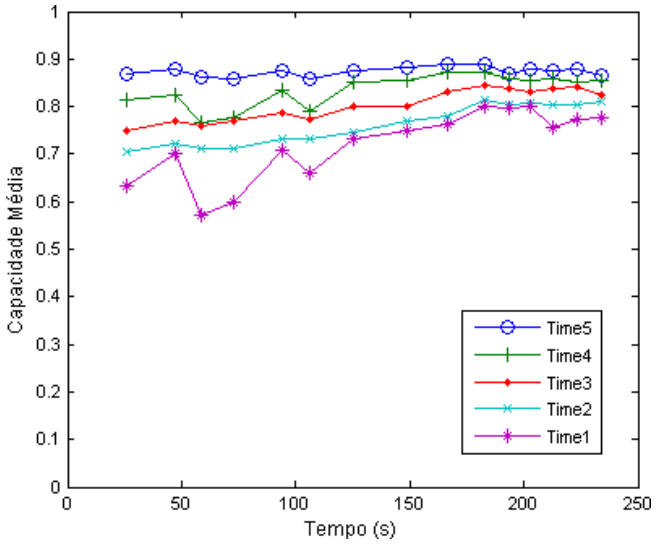

Figura 4.15: Capacidade média utilizada no tempo com tolerância 0.04.

Nas Figuras 4.15, 4.16 e 4.17 mostram-se os resultados obtidos depois de uma execução do PCS com uma tolerância de 0.04 sobre os limiares. Na Tabela 4.4 mostra-se a quantidade de tarefas presentes no sistema em cada momento do experimento. Os resultados são melhores que os obtidos sem o uso da tolerância dado que a capacidade média utilizada com learned threshold diminui menos (pontos inferiores na Figura 4.15) depois de um cálculo de limiares baseado somente na distribuição de capacidades.

4.3.3 Pareto-Eficiência

Uma característica importante que deve possuir todo protocolo de negociação é a pareto-eficiência [Wooldridge (2002); Kraus (1996)]. É desejável que os resultados de todo protocolo

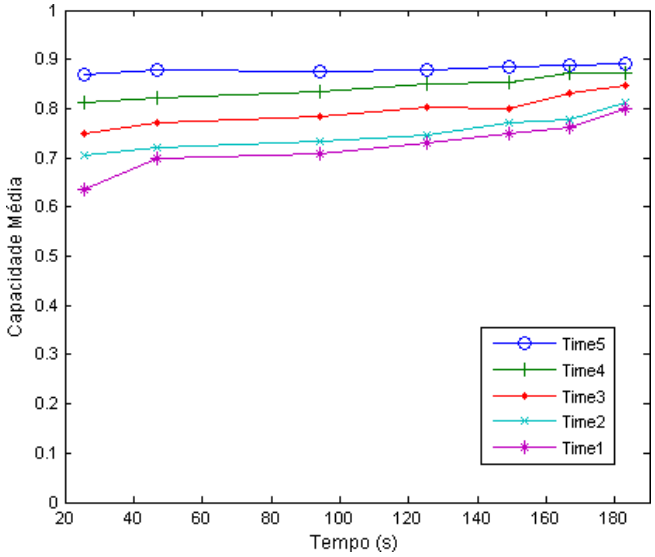

Figura 4.16: Capacidade média no tempo utilizando o cálculo baseado na distribuição de capacidades.

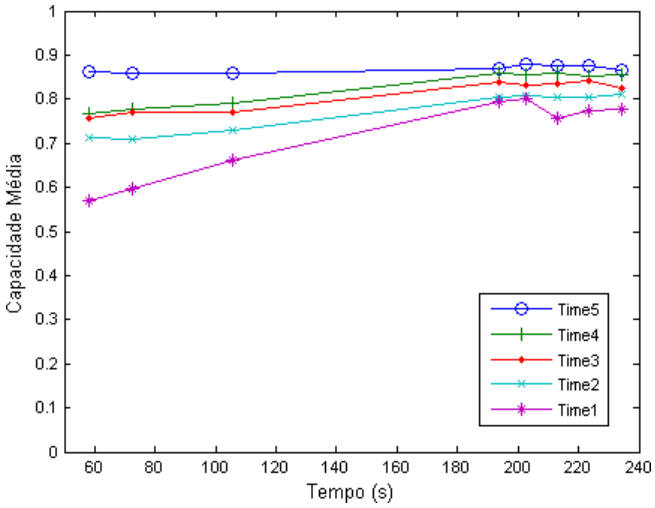

Figura 4.17: Capacidade média no tempo utilizando learned threshold com tolerância 0.04.

sejam ótimos, isto é, que não existam outros possíveis resultados onde ambos negociantes obtêm um ganho maior. Os experimentos mostrados nesta seção têm o objetivo de determinar o nível de pareto-eficiência no PCS.

Na Figura 4.18 mostra-se o ganho obtido pelo PCS comparado com a contratação gulosa tendo em conta o máximo ganho possível em cada teste (soma dos pagamentos recebidos por todas as tarefas dos times). Cada experimento utilizou uma quantidade diferente de robôs e tarefas dada pela proporção ($\mid E \mid / \mid \Theta \mid$) mostrada no eixo x da Figura 4.18. No eixo y mostra-se o ganho obtido. Cada ponto na figura é a média de 10 execuções. Ambos o PCS e a CG utilizaram os mesmos conjuntos de robôs e tarefas. Na Tabela 4.5 mostra-se a percentagem do ganho máximo obtido pelos dois algoritmos.

Nos resultados, o PCS mantém um ganho próximo ao ganho máximo. A CG possui

Tempo (s)	*Time1*	*Time2*	*Time3*	*Time4*	*Time5*	*Total Tarefas*
25.59	40	69	59	72	120	360
46.9	34	63	53	66	114	330
58.26	34	63	53	66	114	330
72.58	34	63	53	66	114	330
94.36	28	57	47	60	108	300
105.87	28	57	47	60	108	300
125.12	22	51	41	54	102	270
149.13	16	45	35	48	96	240
166.85	10	39	29	42	90	210
183.23	4	33	23	36	84	180
193.69	4	33	23	36	84	180
202.59	4	33	23	36	84	180
212.87	4	33	23	36	84	180
223.28	4	33	23	36	84	180
234.22	4	33	23	36	84	180

Tabela 4.4: Número de Tarefas no tempo para o experimento com tolerância 0.04.

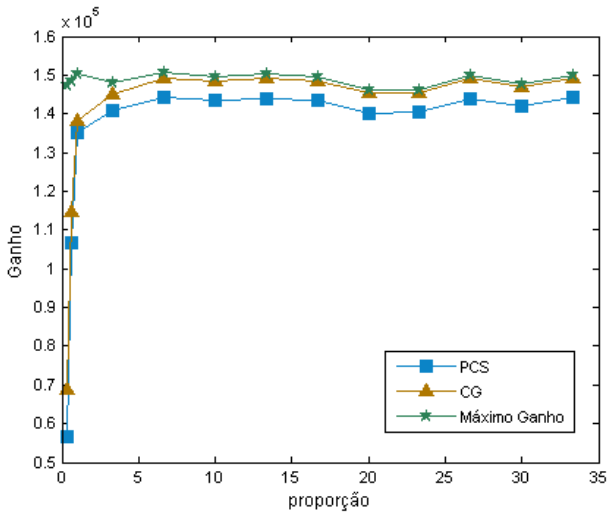

Figura 4.18: Análise pareto-eficiente do PCS.

um ganho maior. 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 4.18) é pequeno em relação à quantidade total de possíveis soluções.

Proporção	*PCS%*	*CG%*
0.3	38.53	46.61
0.7	71.92	77.22
1.0	90.02	91.99
3.3	95.22	98.09
6.7	95.80	98.90
10.0	95.93	99.14
13.3	95.78	99.26
16.7	96.01	99.37
20.0	95.99	99.44
23.3	96.03	99.50
26.7	95.87	99.52
30.0	96.10	99.56
33.3	96.15	99.60

Tabela 4.5: Percentagem do ganho máximo para o análise pareto-eficiente.

4.3.4 Racionalidade Individual

Outra característica desejável num protocolo de negociação é a racionalidade individual. Um protocolo possui racionalidade individual se a utilização do protocolo for benéfico para cada agente, ou seja, cada agente ganha mais utilizando o protocolo do que ganharia não utilizando-o. Dois foram os fatores considerados para medir o nível de racionalidade individual dentro do PCS: concorrência justa e oferta–demanda. Um protocolo de negociação que dê as mesmas oportunidades aos competidores tende a ser benéfico para todos os negociantes. Espera-se que times com mais recursos para competir tenham um ganho maior. Se isso não acontece esses times não terão vantagem em utilizar o protocolo. Além disso, um protocolo onde os preços são determinados pelas leis da oferta e demanda também tende a ser benéfico para os negociantes. Times que ofertem produtos escassos esperarão ganhar mais do que times que ofertem produtos comuns. Se isso não acontece os times que ofertam produtos escassos também não terão vantagem em utilizar o protocolo.

4.3.4.1 Concorrência Justa

Dentro do conjunto de experimentos que medem a concorrência justa, também foram realizados testes que utilizaram limiares globais dentro do PCS, um cálculo de limiares que leva em conta tarefas de outros times como em uma alocação entre agentes cooperativos. Limiares calculados dessa forma permitiriam diminuir a quantidade de mensagens, mas poderiam tornar o protocolo pouco justo, como descrito na Seção 3.8. Os experimentos com limiares globais permitiram analisar a justiça na concorrência com o uso deste tipo de limiares.

Na Figura 4.19 mostra-se a capacidade média utilizada pelos times para 3 relações de robôs e tarefas ($\mid E \mid / \mid \Theta \mid$: 0.5, 1.0 e 2.0). Cada barra na figura é a média de 10 execuções. Utilizou-se 7 times, onde times com maior índice possuem pagamentos médios maiores e estão representados no eixo x da Figura 4.19. No eixo y representa-se a capacidade média utilizada

por cada time para contratar todo o conjunto de tarefas em cada execução do protocolo. Na Tabela 4.6 mostra-se o pagamento médio para as tarefas de cada time nos experimentos. Nas Figuras 4.20, 4.21 e 4.22 mostra-se a capacidade média utilizada para contratar tarefas com pagamentos entre 0 e 2500 em cada uma das execuções nos experimentos da Figura 4.19. No eixo x mostra-se o pagamento médio recebido pelas tarefas de um time em cada execução.

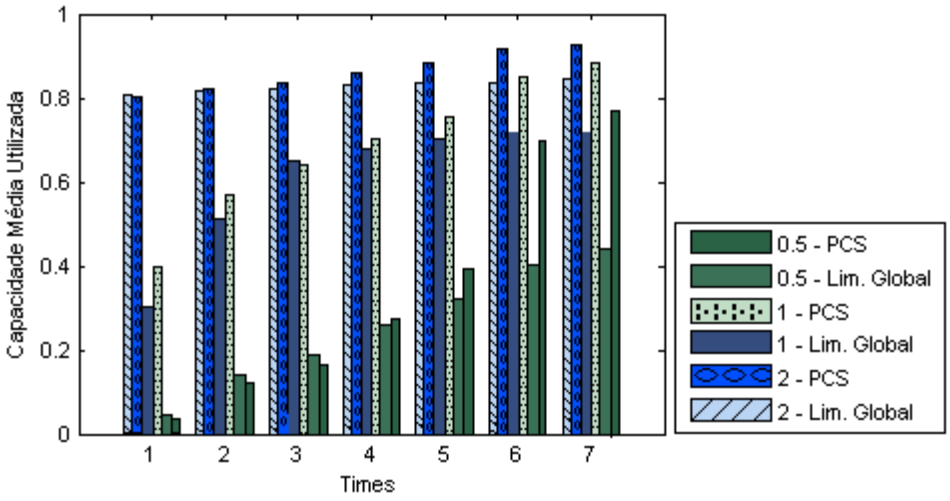

Figura 4.19: Concorrência Justa no PCS.

	Time1	*Time2*	*Time3*	*Time4*	*Time5*	*Time6*	*Time7*
Pagamento médio ($)	327.43	568.93	890.89	1215.26	1536.62	1858.48	2257.30

Tabela 4.6: Pagamento médio pelas tarefas de cada time.

Na Figura 4.19 os times com um pagamento médio maior utilizaram uma capacidade média maior para contratar robôs, o que mostra uma concorrência justa dentro do PCS. Na Figura 4.19 também mostra-se a capacidade média utilizada pelos times com um cálculo de limiares 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. Os times com um pagamento médio menor utilizaram capacidades médias maiores e menores do que a capacidade média utilizada no PCS. Portanto a utilização de limiares globais produz uma concorrência menos justa que outorga um ganho menor a times que se prepararam melhor para concorrer e em muitos casos diminuem a capacidade média utilizada por todos os times.

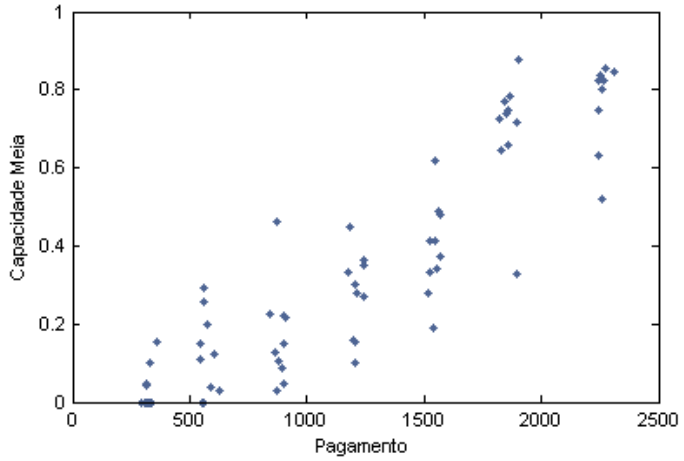

Figura 4.20: Concorrência Justa no PCS utilizando uma proporção $\mid E \mid / \mid \Theta \mid = 0.5$.

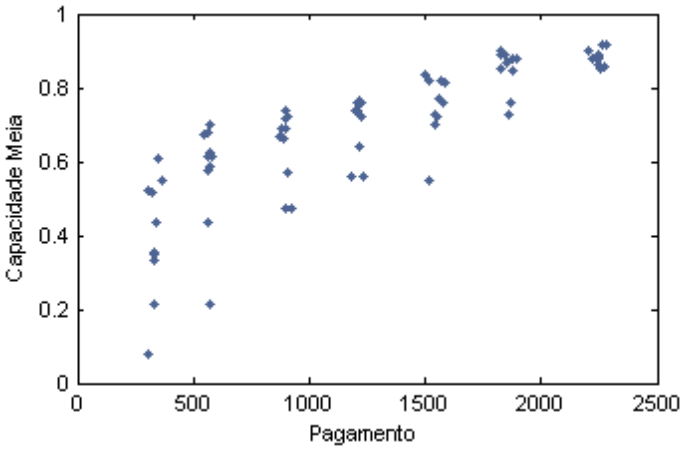

Figura 4.21: Concorrência Justa no PCS utilizando uma proporção $\mid E \mid / \mid \Theta \mid = 1.0$.

4.3.4.2 Preço

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 robôs e tarefas. Na Figura 4.23 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.

Na Figura 4.23, quando a quantidade de tarefas é grande em relação à quantidade de robôs o aluguel médio é maior. O aluguel médio diminui quando a quantidade de tarefas é pequena em relação à quantidade de robôs no sistema. Portanto o preço é determinado pela oferta e a demanda.

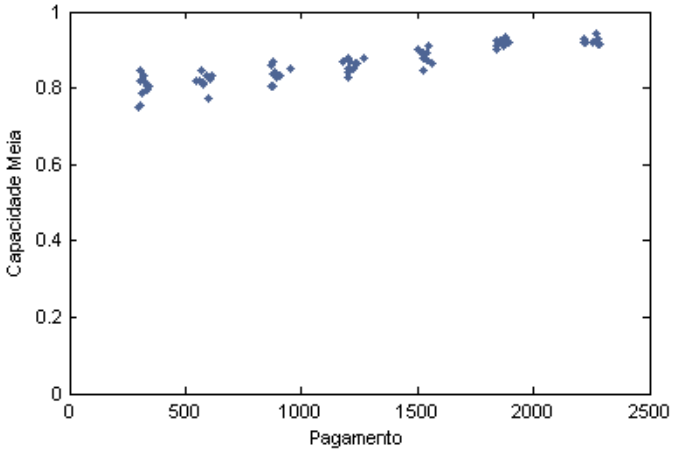

Figura 4.22: Concorrência Justa no PCS utilizando uma proporção $\mid E \mid / \mid \Theta \mid = 2.0$.

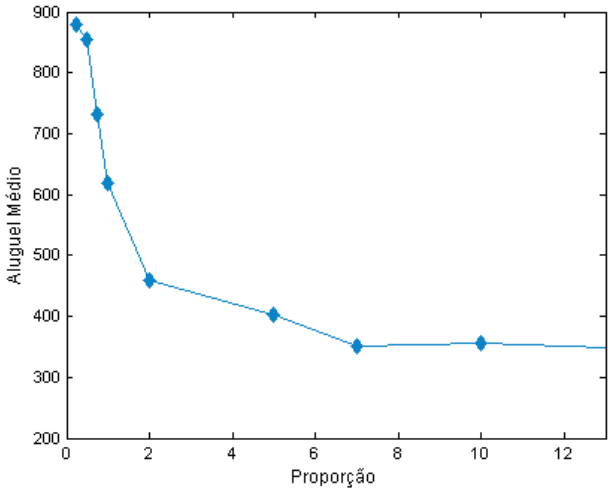

Figura 4.23: Aluguel médio para diferentes proporções de robôs e tarefas.

4.3.5 Simplicidade e Distribuição

Características avaliadas de forma não quantitativa foram a simplicidade e distribuição. Intuitivamente, pode se dizer que o protocolo é simples dado que determinar a estratégia ótima é fácil para cada agente. Todos os algoritmos dentro do PCS são tratáveis, portanto cada agente pode determinar a estratégia ótima utilizando uma quantidade de recursos aceitável. O protocolo também é distribuído dado que os contratos são formados por meio de uma interação agente – agente. Não existem terceiras partes ou pontos de controle na negociação. Dessa forma, as tarefas são contratadas sem a existência de únicos pontos de falha. Isso também contribui para minimizar a comunicação e, dessa forma, tornar o protocolo mais escalável.

4.4 Considerações Finais

Neste capítulo foram apresentados e analisados os resultados dos experimentos realizados com o PCS. Os experimentos mediram a escalabilidade do PCS com um cálculo de limiares baseado na distribuição de capacidades e baseado na concorrência. Eles mediram também a pareto-eficiência, adaptabilidade e racionalidade individual do protocolo, essa última em função da concorrência justa, da oferta e da demanda.

O protocolo mostrou ser escalável ao utilizar uma quantidade de mensagens constante independente do número de robôs presentes no sistema. Além disso, a utilização de limiares aprendidos permitiu diminuir a quantidade de mensagens utilizadas por cada time. O PCS também mostrou ser pareto eficiente ao obter um ganho próximo ao máximo ganho possível em cada experimento. Com relação aos experimentos que mediram a flexibilidade, o PCS adaptou-se bem a mudanças no número de tarefas e agentes, aproveitando os recursos disponíveis. Além disso, o PCS mostrou ser racionalmente individual dado que permitiu que times com melhores condições para concorrer possam obter maior ganho e dado que permitiu também uma determinação de preços justos baseados na oferta e demanda.

Capítulo 5

Conclusões e Perspectivas Futuras

O problema abordado nesta dissertação foi a contratação de agentes em grupos compostos por uma grande quantidade de indivíduos (swarms). A contratação de agentes é um processo complexo onde os agentes envolvidos precisam negociar para determinar as condições da cooperação (preços, esforço, penalidades, etc). Em grandes grupos, essa complexidade aumenta pois é difícil determinar todas as possibilidades de contratação. Especificamente, não é viável possuir informações de cada agente no sistema, ou seja, não é viável possuir informações de cada possível agente contratado. Para solucionar este problema foi proposto um protocolo de contratação (PCS), projetado especialmente para trabalhar com grupos de agentes em grande escala. As contribuições do PCS para a solução do problema de contratação em swarms são:

- Uma interação escalável entre agentes. Cada agente interage com um grupo reduzido de outros agentes dentro do swarm. A quantidade de agentes dentro do grupo de interação não depende da quantidade total de agentes no sistema. Com isso, a comunicação utilizada na contratação não muda com aumentos ou diminuições no tamanho do grupo. A comunicação utilizada depende apenas da quantidade de tarefas que precisam ser alocadas. Com isso, a comunicação utilizada permanece em níveis baixos mesmo em grupos compostos por quantidades extremas de agentes. Além disso, os agentes envolvidos na contratação não precisam realizar processamentos complexos. Os agentes baseiam suas decisões tanto em informação global quanto individual, mas não utilizam informação de cada elemento no sistema, dado que isso obrigaria o protocolo a utilizar um alto nível de processamento no caso de grandes grupos de agentes.

- Um processo eficiente e justo de contratação. O PCS está projetado para encontrar agentes bem capacitados para realizar as tarefas. Além disso, as tarefas que têm preferência são as tarefas com maior pagamento. Essas duas características produzem uma contratação eficiente próxima à ótima. Também, no PCS, times que estão mais preparados para concorrer obtêm maior ganho que times menos preparados, existindo assim justiça na concorrência. Os preços no PCS são também justos. Um agente pode atuar racionalmente considerando como justos os preços regidos pela oferta e demanda. Os preços calculados no PCS são determinados por leilões entre agentes que ofertam e agen-

tes que demandam serviços. Portanto, o preço calculado para a realização do serviço depende da quantidade de agentes nos dois conjuntos.

- Uma contratação mista entre agentes com interesses comuns e próprios. A contratação dentro do PCS integra a alocação de tarefas entre agentes totalmente cooperativos e a contratação entre agentes que possuem interesses próprios. Os agentes possuem um mesmo comportamento que lhes permite interagir tanto com agentes do mesmo time como com agentes de outros times, ou seja, o protocolo geral de contratação é o mesmo para os dois tipos de agentes. Para conseguir isso, a contratação dentro de um time no PCS adequou-se aos mecanismos utilizados na interação entre agentes com interesses individuais, descartando processos não aplicáveis dentro de uma alocação simples de tarefas.

- Um cálculo de limiares utilizando informação global imprecisa. Como aos times é permitido não revelar algumas informações, a informação global que se possui, que permitiria uma contratação eficiente e escalável, não é exata. O PCS permite uma contratação escalável ainda nessas condições utilizando limiares aprendidos em contratações prévias (*learned threshold*). Os limiares aprendidos permitem que um time não incremente a comunicação utilizada, tentando contratar agentes bem capacitados, que serão contratados finalmente por times que têm melhores condições para competir. Limiares calculados com informação global inexata podem ser utilizados também em alocações simples de tarefas em grandes grupos cooperativos. Isso pode ser particularmente útil quando não se possui toda a informação global necessária ou quando o cálculo de limiares é complicado.

Além disso, o PCS permite realizar uma contratação entre agentes com interesses individuais considerando problemas próprios da interação entre agentes não totalmente cooperativos. Em sistemas compostos por agentes com interesses individuais não é possível considerar que todos os agentes estão dispostos a cooperar para atingir o bem comum. Nesses sistemas, como os agentes cooperam somente para atingir seus próprios objetivos, aspectos tais como: incentivos, falta de honestidade, não cumprimento de contratos, informação privada, identidades falsas, etc. devem ser tratados. O PCS foi projetado considerando alguns desses problemas, especificamente: incentivos e informação privada.

Num sistema de agentes com interesses próprios, os times desejam manter privadas certas informações. Custos e pagamentos pela realização de tarefas são dois tipos de informação que times de agentes devem manter como informação confidencial. O PCS permite que os times mantenham essa informação privada, utilizando estimativas de custos e pagamentos de forma tal que os agentes não precisam revelar essa informação para conseguir cooperar. Além disso, o PCS utiliza tipos de leilão que não expõem a informação privada dos agentes, especificamente, o leilão inglês e o holandês. Esses mesmos leilões permitem a determinação de preços que servem de incentivo para que agentes com interesses próprios cooperem.

5.1 Discussão

O PCS foi testado numa tarefa de transporte de mercadorias implementada em um simulador multi-robô chamado MuRoS. Os testes compararam o PCS com uma implementação gulosa da contratação de agentes, em swarms de diferentes tamanhos. Esses experimentos demonstraram a escalabilidade do protocolo. Os resultados mostram uma quantidade de mensagens constante para o PCS independente do tamanho do swarm e um ganho próximo do ganho máximo. Apesar de utilizar uma quantidade de mensagens muito inferior à quantidade utilizada pela contratação gulosa, o PCS obteve um ganho bastante similar.

Também foram realizados testes específicos com o learned threshold. O seu uso permitiu diminuir a quantidade de mensagens especialmente quando existe concorrência. Não obstante, o ganho obtido com a utilização deste limiar ainda que próximo foi um pouco inferior comparado com o ganho obtido sem utilizá-lo. Essa diminuição no ganho para o learned threshold é causado pelo seguinte comportamento observado dentro do PCS: considere $T1_{min}$ como o mínimo limiar utilizado numa contratação com cálculo de limiares baseados na distribuição de capacidades. Se $T1_{min}$ fosse utilizado numa contratação com o learned threshold, os agentes contratados poderiam ter capacidades menores que $T1_{min}$, uma vez que esse limiar pode ser diminuído durante o processo de descoberta. As contratações de agentes que possuem capacidades menores do que $T1_{min}$ (que é o mínimo limiar utilizado nas contratações com limiares baseados na distribuição de capacidades) provocam a diminuição do ganho com a utilização de learned thresholds. Esse efeito pode ser diminuído com a utilização de uma tolerância que permita utilizar limiares um pouco maiores do que os aprendidos, como limiares iniciais para os tokens. Dessa forma, algumas vezes seriam contratados agentes com capacidades maiores e outras vezes agentes com capacidades menores do que os limiares aprendidos, mas na média, os tokens utilizariam uma capacidade igual ao limiar aprendido, evitando a diminuição no ganho. Os testes utilizaram uma tolerância calculada empiricamente que permitiu obter bons resultados. No entanto, é necessário estabelecer um cálculo específico para a tolerância, de maneira que seja possível encontrar valores adequados a cada situação.

O PCS também foi submetido a testes de adaptabilidade a fim de se determinar a capacidade de adaptação em ambientes dinâmicos. Em geral, o PCS mostrou uma boa adaptação a mudanças no ambiente: ele utilizou uma capacidade média maior quando se dispunha de maior capacidade no swarm e uma capacidade média menor quando a quantidade de agentes bem capacitados foi menor. Nos testes, o PCS utilizou somente um limiar baseado na distribuição de capacidades quando a quantidade de tarefas para todos os times mudava e limiares baseados na concorrência quando as condições não mudavam ou quando apenas o número de agentes era alterado. Atualmente, o PCS pode adaptar limiares baseados na concorrência quando a quantidade de tarefas dos times aumenta, utilizando para isso o método de diminuição de T_{min} descrito no protocolo (Seção 3.4). No entanto, o PCS não possui um método para adaptar os limiares baseados na concorrência quando a quantidade de tarefas dos times diminui, em outras palavras, o PCS não consegue aumentar os limiares baseados na concorrência para aproveitar agentes melhor capacitados quando se conta com maior quantidade

deles. Portanto, o PCS precisa da definição de um método para adaptar limiares baseados na concorrência quando a quantidade de tarefas dos times diminui. Como discutido na Seção 4.3.2, um possível método para se conseguir isso é fazer percorrer os tokens com limiares baseados na distribuição de capacidades um número de passos que estariam em função da mudança no número de tarefas.

Outros tipos de características, como a pareto-eficiência e a racionalidade individual, são desejáveis em todo protocolo de contratação e também foram avaliadas no PCS. Os resultados foram muito satisfatórios. O PCS mostrou ser pareto-eficiente dado que a quantidade de possíveis melhorias nos contratos foi pequena em relação à quantidade total de possíveis soluções. A contratação também se mostrou individualmente racional. Um agente, atuando racionalmente, preferirá utilizar um protocolo que seja justo. A contratação foi justa no cálculo de preços, ao calculá-los em função da oferta e a demanda. A contratação também foi justa na concorrência ao dar um maior ganho a times melhor preparados para concorrer.

5.2 Trabalhos Futuros

Existem muitas possíveis direções futuras de pesquisa referentes ao PCS. Por exemplo, o PCS não considera temas como: decomposição e dependência de/entre tarefas, execução simultânea de múltiplas tarefas, penalidades, níveis de esforço, falsidade de identidades, etc. Elementos que estão presentes em outros protocolos de contratação e que poderiam ser inclusos dentro do PCS.

Outra possível melhoria para o PCS se refere ao cálculo de limiares baseados na concorrência. O cálculo desses limiares é ainda muito simples. Ele está baseado somente nos valores máximos e mínimos atingidos pelos limiares. O cálculo deste tipo de limiar pode ser melhorado com a utilização de técnicas mais sofisticadas como séries temporais ou com a utilização de outros parâmetros estatísticos (média, desvio padrão). O emprego de técnicas que utilizem informação histórica de limiares além dos limiares utilizados na última execução do protocolo seria muito útil para o PCS.

Especificamente, uma linha de pesquisa interessante se refere à utilização e a avaliação do learned threshold como método para corrigir limiares calculados na alocação de tarefas em grandes grupos cooperativos. Como foi dito na Seção 3.7.2, em ambientes onde o cálculo de limiares é complexo (não se possui informação global ou a distribuição de capacidades muda freqüentemente) o learned threshold pode ajudar a calcular bons limiares. Muitas vezes a informação global que se tem é inexata ou, ainda que se possua toda a informação, a criação de uma fórmula que contemple todos os casos é complicado. Por exemplo, a criação de fórmulas para calcular limiares que sejam adequadas para todo tipo de distribuição de capacidades é uma tarefa não trivial. Nesses casos a utilização de informação obtida em processos prévios pode facilitar o cálculo de limiares.

Além disso, um trabalho adicional que poderia ser feito é a analise formal dos aspectos computacionais do PCS de forma a analisar melhor a sua distribuição e simplicidade. Mesmo que intuitivamente o PCS pareça ser simples e possuir um bom nível de distribuição, um estudo

mais profundo considerando, por exemplo, o paralelismo ou a complexidade dos algoritmos seria recomendável.

Por fim, um outro estudo interessante seria a utilização do PCS em outras tarefas que apresentem diferentes características como outras distribuições de capacidades, distribuições de capacidades dinâmicas (distribuições que mudam durante a execução das tarefas) ou o estudo da aplicabilidade do protocolo, por exemplo, na contratação de agentes na internet. Também, dado que os testes realizados nesta dissertação foram todas simulações, seria interessante a realização de implementações reais para swarms de robôs, onde poderia se observar o comportamento do protocolo em ambientes físicos.

Referências Bibliográficas

Anthony, P.; Hall, W.; Dang, V. e Jennings, N. (2001). Autonomous agents for participating in multiple on-line auctions. In *Proccedings of The International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 54–64.

Babanov, A.; Collins, J. e 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.

Bonabeau, E.; Dorigo, M. e Théraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press Inc, USA.

Bonabeau, E. e Théraulaz, G. (2000). Swarm smarts. *Scientific American*, pp. 73–79.

Botelho, S. e Alami, R. (1999). M+ : a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *Proccedings of The IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1234–1239.

Chaimowicz, L.; Campos, M. e Kumar, V. (2002). Dynamic role assignment for cooperative robots. In *Proccedings of The IEEE International Conference on Robotics and Automation (ICRA)*, pp. 293–298.

Chaimowicz, L.; Michael, N. e Kumar, V. (2005). Controlling swarms of robots using interpolated implicit functions. In *Proccedings of The IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2498–2503.

Cicirello, V. e Smith, S. (2001). Wasp nests for self-configurable factories. In *Proccedings of The International Conference on Autonomous Agents (AGENTS'01)*, pp. 473–480.

Collins, J.; Ketter, W.; Gini, M. e 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.

Correll, N.; Rutishauser, S. e Martinoli, A. (2006). Comparing coordination schemes for miniature robotic swarms: A case study in boundary coverage of regular structures. In *Proceedings of The International Symposium on Experimental Robotics (ISER)*.

Ferreira, P.; Oliveira, D. e Bazzan, A. L. C. (2005). A swarm based approach to adapt the structural dimension of agents organizations. *Journal of the Brazilian Computer Society*, 11(1):63–73.

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

Gaston, M. e desJardins, M. (2005). Social network structures and their impact on multi-agent systems dynamics. In *Proccedings of The International Conference of The Florida Artificial Intelligence Reseach Society (FLAIRS)*, pp. 32–37.

Gerkey, B. e Mataric, M. (2002). Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768.

Gerkey, B. e Mataric, M. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954.

Greenwald, A. e Kephart, J. (1999). Shopbots and pricebots. In *Proccedings of The Workshop on Agent Mediated Electronic Commerce in IJCAI (AMEC'99)*, pp. 1–23.

Hsieh, M. A. e Kumar, V. (2006). Pattern generation with multiple robots. In *Proccedings of The IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2448–2447.

Ishida, T.; Gasser, L. e Nakashima, H., editores (2005). *Massively Multi-Agent Systems I, First International Workshop, MMAS 2004, Kyoto, Japan, December 10-11, 2004, Revised Selected and Invited Papers*, volume 3446 of *Lecture Notes in Computer Science*. Springer.

Jang, M.-W. e 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)*, pp. 19–33.

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

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

Kraus, S. (2001). Automated negotiation and decision making in multiagent environments. *Lecture Notes in Artificial Intelligence*, pp. 150–172.

Markopoulos, P. e Kephart, J. (2002). How valuable are shopbots? In *Proccedings of The International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1009–1016.

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

McLurkin, J. e Smith, J. (2004). Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots. In *Proceedings of The Distributed Autonomous Robotic Systems Conference (DARS)*, pp. 381–390.

Parker, L. (1998). Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240.

Pimenta, L. C. A.; Mendes, M. L.; Mesquita, R. C. e Pereira, G. A. S. (2006). Fluids, particles, and multiple robots in electrostatic fields. In *Proceedings of The IEEE Conference on Electromagnetic Field Computation (CEFC)*, volume 4, p. 309.

Rothkopf, M.; Teisberg, T. e Kahn, E. (1990). Why are vickrey auctions rare? *Journal of Political Economy*, 98(1):94–109.

Sandholm, T. (1999). Distributed rational decision making. In WeiB, G., editor, *Multiagent systems: a modern approach to distributed artificial intelligence*, pp. 201–258. MIT Press, Cambridge, MA, USA.

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)*, pp. 256–262.

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

Sandholm, T. W. e 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)*, pp. 328–335.

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

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

Shehory, O.; Kraus, S. e Yadgar, O. (1998). Emergent cooperative goal-satisfaction in large scale automated-agent systems. *Artificial Intelligence Journal*, 101(1):1–55.

Smith, R. (1977). The contract net: A formalism for the control of distributed problem solving. In *Proccedings of The International Joint Conference on Artificial Intelligence (IJCAI)*, p. 472.

Smith, R. (1980a). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113.

Smith, R. (1980b). A framework for distributed problem solving. *UMI Research Press.*

Smith, R. e Davis, R. (1981). Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):61–70.

Tesfatsion, L. e Judd, K. L. (2006). *Handbook of Computational Economics*, volume 2. North Holland.

Tosic, P. e Agha, G. (2004). Maximal clique based distributed group formation for task allocation in large-scale multi-agent systems. In *Proccedings of The International Workshop on Massively Multi-Agent Systems (MMAS'04).*

Turner, P. e Jennings, N. (2000). Improving the scalability of multi-agent systems. In *Proccedings of The Workshop on Infrastructure for Scalable Multi-Agent Systems in Autonomous Agents 2000*, pp. 246–262.

Valckenaers, P.; Kollingbaum, M.; Brussel, H. V.; Bochmann, O. e Zamfirescu, C. (2001). The design of multi-agent coordination and control systems using stigmergy. In *Proccedings of The International Workshop on Emergent Synthesis(IWES'01).*

Vickrey, W. (1961). Counterespeculation, auctions and competitive sealed tenders. *Journal of Finance*, 18:8–37.

Vlassis, N. (2003). *A Concise Introduction to Multiagent Systems and Distributed AI.* Informatics Institute, University of Amsterdam.

Wooldridge, M. J. (2002). *An Introduction to Multi-Agent Systems.* John Wiley & Sons Ltd.

Xu, Y.; Lewis, M.; Sycara, K. e Scerri, P. (2004). Information sharing in very large teams. In *Proccedings of The Workshop on Challenges in the Coordination of Large Scale Multi-agent Systems in AAMAS'04 (LSMAS'04).*

Xu, Y.; Liao, E.; Scerri, P.; Yu, B.; Lewis, M. e Sycara, K. (2005a). Towards flexible coordination of large scale multi-agent teams. In *textbook Coordination of Large Scale Multiagent Systems*, pp. 287–309. Springer US.

Xu, Y.; Scerri, P.; Sycara, K. e Lewis, M. (2006). Comparing market and token-based coordination. In *Proccedings of The International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1113–1115.

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