

Alocação de tarefas em sistemas multi-robóticos

Antonio Luiz Basile
Prof. Flávio Soares Corrêa da Silva
IME - USP

25 de junho de 2004

1 Introdução

Na última década houve intensa pesquisa em sistemas multi-robóticos, seja por serem estes uma extensão natural de sistemas multi-agentes, seja pela demanda por sistemas capazes de realizar tarefas perigosas (recolhimento de lixo atômico, vigilância, exploração planetária, busca e resgate, etc) bem como realizar tarefas fatigantes aos humanos (manutenção industrial, manufatura, etc) [Par00, DJMW96, CFK95, INS01, Par98, GM00, GVS⁺01, GM01, GM02a, GM02b, GM03]. O uso de vários robôs ao invés de um robô monolítico capaz de se encarregar de tudo parece mais adequado a muitas destas tarefas, até porque muitas delas são atualmente realizadas por grupos de humanos. Algumas alternativas para um projeto de sistemas multi-robóticos são:

- Se o sistema será composto por robôs que cooperam para a realização da tarefa ou se competem entre si.
- Se os robôs realizam coordenadamente a tarefa ou não.
- Se o sistema é homogêneo ou heterogêneo, quanto aos modelos de robôs. No estudo de sistemas multi-robóticos cooperativos heterogêneos, um ponto de grande relevância para o sucesso na realização das tarefas é a escolha de qual robô deve realizar qual tarefa (ou tarefas) e aos que já encontram-se alocados, se há a necessidade de realocação.

Para incrementar a eficiência do sistema, muitos pesquisadores utilizam alguma métrica para efetuar a escolha do robô para determinada tarefa em conjunto a um esquema de leilões inspirado no contract net protocol [Smi80, San93, Zlo04]. A métrica mais usualmente utilizada é alocar-se o robô mais apto, onde por "mais apto" entende-se o robô que possua as características mais adequadas na realização da tarefa, em relação a seus pares. Esta característica do robô pode ser desde uma aptidão nata (se pode ou não locomover-se, qual sua velocidade de locomoção, qual o custo para efetuar a locomoção, comunicação, etc) até seu status corrente (a que distância encontra-se o robô do alvo, se está ou não impossibilitado de sair de uma sala, etc). Este documento reflete o estudo da tese de doutorado de Brian Paul Gerkey defendida na University of Southern California em agosto de 2003, bem como de seus desdobramentos bibliográficos [Ger03]. A seguir descrevo os problemas da atribuição ótima

(OAP-Optimal Assignment Problem), bem como da alocação de tarefas para sistemas multi-robóticos (MRTA - Multi-Robot Task Allocation) e as diversas abordagens dadas a este problema por algumas áreas como pesquisa operacional, teoria dos grafos, otimização combinatória, etc. Feito isto apresento a análise formal e taxonomia para alocação de tarefas em sistemas multi-robóticos proposta por na tese citada. Em seguida faço um estudo detalhado do algoritmo de Munkre-Kuhn (uma versão com pesos nas arestas do algoritmo húngaro) através de sua descrição, simulação e implementação. O objetivo principal deste estudo é servir como base para futura pesquisa para uma forma de avaliação (métrica) pouco adotada em alocação de tarefas para sistemas multi-robóticos heterogêneos [Bot98, SB99] bem como pesquisar a viabilidade ou não do uso do plano que cada robô efetua para realizar dada tarefa, como métrica, ou seja, comparar os planos dos robôs entre si, escolhendo o robô que parece ter o melhor plano, discriminando os pontos fortes e fracos de tal abordagem se comparada com outras abordagens. A escolha de uma linguagem de programação mais adequada para o domínio escolhido [GL99, GLLS02], e a justificativa de tal escolha, também farão parte deste estudo futuro, bem como a descrição dos tipos de leilões mais adotados em sistemas multi-robóticos [San00, San98, Smi80, San93, Zlo04].

2 Problema

O problema da alocação de tarefas para sistemas multi-robóticos (MRTA - Multi-robot task allocation) pode ser formulado como um problema onde, dados alguns recursos, algumas tarefas e uma métrica de performance, a meta do sistema é maximizar a performance nas tarefas, sujeito às restrições impostas pelos recursos [Ger03]. Este tipo de problema foi estudado por diversas áreas, cada uma com sua visão particular do problema. Dentre estas áreas estão a pesquisa operacional, a teoria dos grafos, otimização combinatória, scheduling e fluxo em redes. A seguir serão apresentadas duas possíveis definições para os problemas da atribuição ótima (OAP - optimal assignment problem) e da alocação de tarefas em sistemas multi-robóticos (MRTA), bem como apresentar a função utilidade, para que possamos, então, resumidamente descrever a forma como cada das áreas citadas acima lida com estes problemas.

2.1 Definições preliminares

Definição 2.1.1 (Problema da atribuição ótima (OAP)) *Sejam m trabalhadores e n trabalhos, onde cada trabalhador pode ser associado a apenas um trabalho e cada trabalho requer apenas um trabalhador. Os trabalhadores têm suas competências em relação a cada um dos trabalhos associadas a valores não negativos (se um trabalhador não puder executar determinado trabalho, terá a si associado zero como valor). Os trabalhos podem ou não estar associados a pesos. A meta é atribuir a cada trabalhador um trabalho visando maximizar a performance total esperada (levando em conta a prioridade do trabalho).*

Definição 2.1.2 (Alocação de tarefas em sist. multi-robóticos (MRTA)) *Sejam m robôs e n tarefas, onde cada robô pode ser associado a apenas uma tarefa e cada tarefa requer apenas um robô. Os robôs têm suas competências em relação a cada uma das tarefas associadas a valores não negativos (se um robô*

não puder executar determinada tarefa, terá a si associado zero como valor). As tarefas podem ou não estar associadas a pesos. A meta é atribuir a cada robô uma tarefa visando maximizar a performance total esperada (levando em conta a prioridade da tarefa).

2.1.1 Utilidade

A utilidade (ou custo) é uma estimativa que cada indivíduo faz sobre seu próprio desempenho na realização de determinada tarefa ou ação (uma alternativa aqui é o indivíduo receber o valor estimado de terceiros, ou seja, externamente). Ou seja, trata-se de uma métrica, que no caso do MRTA pode variar de uma simples avaliação baseada nos sensores do robô, até uma complexa avaliação baseada em planejamento como o exposto no projeto M^+ [Bot98, SB99]. Para a utilidade devemos levar em conta a qualidade esperada para a execução da tarefa, bem como o custo esperado de recursos. Então podemos definir a função utilidade U_{RT} de um robô R para uma tarefa T , em função do custo esperado desta tarefa para este robô (C_{RT}) e da qualidade esperada na realização desta tarefa por este robô (Q_{RT}), ou seja:

$U_{RT} = Q_{RT} - C_{RT}$, se $Q_{RT} > C_{RT}$ e R é capaz de realizar T

ou

$U_{RT} = 0$, em caso contrário.

2.2 Pesquisa operacional

A pesquisa operacional, mais especificamente a programação linear também contribui para a solução de problemas de maximização e para problemas de minimização. Vamos a algumas definições preliminares.

Definição 2.2.1 (Problema do máximo para um programa linear) *Dada uma matriz A $m \times n$, um n -vetor b e um m -vetor c , encontre um m -vetor não negativo x , tal que xc é um máximo sujeito a $xA \leq b$.*

Se existir um vetor x que satisfaça as restrições, então o problema é dito factível e x é uma solução factível. Uma solução factível é ótima se nenhuma outra solução produzir um valor maior para a função a ser maximizada. Teremos um programa linear inteiro (ILP-integer linear program) se os elementos do vetor solução tiverem de ser inteiros. O problema OAP é um tipo especial de ILP, pois exige uma estrutura que facilita sua solução (o problema 0-1 ILP é NP-difícil).

2.2.1 MRTA vista como um programa linear

Para a programação linear, o problema MRTA pode ser assim descrito: Encontre n^2 inteiros não negativos α_{ij} que maximize

$$\sum_{i=1}^n \sum_{j=1}^n \alpha_{ij} U_{ij} w_j$$

sujeito a

$$\sum_{i=1}^n \alpha_{ij} = 1, \quad 1 \leq j \leq n$$

$$\sum_{j=1}^n \alpha_{ij} = 1, \quad 1 \leq i \leq n$$

ou seja, um robô só realiza uma tarefa por vez e uma tarefa só precisa de um robô para ser realizada. Para resolver este problema utilizamos o algoritmo simplex, que apesar de no pior caso ser exponencial para o tamanho da entrada, no caso médio, para certas classes de problemas é polinomial.

2.3 Teoria dos grafos

A teoria dos grafos modela o problema OAP (e consequentemente o problema MRTA) através de um emparelhamento perfeito num grafo bipartido com pesos em suas arestas. Abaixo algumas definições preliminares [bon77]:

Definição 2.3.1 (Grafo bipartido) *Grafo bipartido é aquele cujo conjunto de vértices pode ser particionado em dois subconjuntos X e Y , tal que cada aresta tem uma terminação em X e outra em Y . Tal partição (X, Y) é chamada uma bipartição do grafo.*

Definição 2.3.2 (Emparelhamento) *Um subconjunto M de E é chamado emparelhamento num grafo G se seus elementos são links sem que existam dois que sejam adjacentes em G .*

Definição 2.3.3 (Emparelhamento Perfeito) *Um emparelhamento M satura um vértice v se alguma aresta de M incide em v . Se cada vértice do grafo G for saturado, o emparelhamento M é um emparelhamento perfeito.*

Definição 2.3.4 (Caminho E-alternante e caminho E-aumentador) *Se M é um emparelhamento em G , um caminho E-alternante em G é um caminho cujos vértices estão alternadamente em $E \setminus M$ e M . Um caminho E-aumentador é um caminho E-alternante cuja origem e término não são saturados.*

A modelagem de OAP na teoria dos grafos pode ser assim descrita. Constrói-se um grafo bipartido G com bipartição (X, Y) , onde $X = \{x_1, x_2, \dots, x_n\}$, $Y = \{y_1, y_2, \dots, y_n\}$, e uma aresta $x_i y_j$ tem peso $w_{ij} = w(x_i y_j)$, como a qualificação do trabalhador X_i para o trabalho Y_j . O problema consiste em determinarmos um emparelhamento perfeito de máximo peso num grafo com pesos [bon77]. Para este problema a teoria dos grafos possui o algoritmo de Munkre-Kuhn (também conhecido como algoritmo húngaro). Na seção 4 veremos detalhadamente este algoritmo e no apêndice está minha implementação (em prolog) para este algoritmo.

2.4 Otimização combinatória

Problemas tais como OAP e MRTA podem ser abordados através da clássica teoria dos subconjuntos independentes em otimização combinatória, ou seja, como um problema de maximização sobre um sistema de subconjuntos. Abaixo são apresentadas três definições importantes para iniciar o estudo de MRTA como um problema em otimização combinatória: subconjuntos independentes, o problema da maximização sobre um conjunto independente e matróides [pap82].

Definição 2.4.1 (Sistema de Subconjuntos) *Um sistema de subconjuntos $S = \{E, I\}$ é um conjunto finito E junto a uma coleção I de subconjuntos de E que está fechado sob inclusão (também chamada propriedade hereditária), ou seja, se $A \in I$ e $A' \subseteq A$, então $A' \in I$. Os elementos de I são chamados independentes.*

Ou seja, qualquer subconjunto de um conjunto independente é também um conjunto independente.

Definição 2.4.2 (Maximização sobre um sist de subconjuntos) *O problema da otimização combinatória associado ao sistema de subconjuntos (E, I) pode ser assim definido: dado um peso $w(e) \geq 0$ para cada $e \in E$, encontre um subconjunto independente que possua o maior peso total.*

Na busca por um algoritmo para este problema, nota-se que não adianta muito representarmos I como todos os subconjuntos independentes de E , pois neste caso teríamos mais que $2^{|E|}$ subconjuntos contidos em I , o que poderia ser muito ineficiente. Ao invés disto podemos pensar num algoritmo onde dado um subconjunto C de E decide se $C \in I$. Tal algoritmo funciona como um espécie de oráculo que possui como tarefa avaliar se uma solução proposta é ou não factível. No caso do problema de maximização sobre um sistema de subconjuntos vamos estudar o algoritmo guloso que para alguns casos apresenta solução ótima, particularmente para aquelas onde os sistemas de subconjuntos podem ser classificados como matróides.

Definição 2.4.3 (Matróide) *Um sistema de subconjuntos $M = (E, I)$ é um matróide se para $I_p, I_{p+1} \in I$ com $|I_p| = p$ e $|I_{p+1}| = p + 1$ existir um elemento $e \in I_{p+1} \setminus I_p$ tal que $I_p + e \in I$.*

Ou seja, posso crescer o subconjunto independente I_p adicionando a ele um elemento que só esteja em I_{p+1} . Note que para matróides se A é um subconjunto independente de E e I e I' são subconjuntos independentes maximais de A , então $|I'| = |I|$ (um subconjunto independente maximal de um matróide é também chamado de base do matróide). Um sistema de subconjuntos $M = (E, I)$ também pode ser definido como um matróide, se o algoritmo guloso encontra solução ótima para cada uma das instâncias do problema de maximização associadas a M . O nosso problema MRTA, em otimização combinatória, pode ser visto da seguinte maneira [Ger03]. Dado um sistema de subconjuntos (E, F) onde E é o conjunto de todos os pares de robôs-tarefas possíveis e F é o conjunto de todas as atribuições possíveis. Também é dada uma função $u(e)$ que retorna a utilidade de cada par $e \in E$ robô-tarefa. A meta é encontrar uma associação $X \in F$ que maximize a soma das utilidades. Este problema é a intersecção com pesos de dois matróides, cujo resultado não é um matróide. Logo não é esperado que um algoritmo guloso encontre a solução ótima para todas as instâncias deste tipo de problema. É sabido que o algoritmo guloso é 2-competitivo para problemas OAP (ou seja, para qualquer entrada encontra uma solução cujo benefício nunca é menor que a metade do benefício da solução ótima). Para problemas MRTA a performance do algoritmo guloso está em aberto. No entanto, alguns resultados empíricos sugerem que para problemas que exibam alguma estrutura e não apresentem combinações da utilidade patológicas, algoritmos gulosos apresentem bons resultados.

2.5 Scheduling

Dado um conjunto de máquinas, um conjunto de processos e um critério de performance, construir um schedule dos processos para cada máquina tal que a performance seja maximizada. Para o MRTA descrevemos problemas de schedule como:

$$R|| \sum w_j C_j$$

ou seja, o sistema é composto por máquinas paralelas não relacionadas e a performance total é computada como uma soma dos pesos dos tempo de processamento para as tarefas individuais. Neste caso definimos o tempo de processamento C_j como K_{ij} , o custo esperado da execução da tarefa j pelo robô i . Este é um superconjunto do problema com máquinas idênticas paralelas que é *NP*-difícil. Em MRTA podemos relaxar algumas restrições, como incorporar os pesos das tarefas diretamente na estimativa dos custos e atribuir tarefas em de forma discreta no tempo e o problema se reduz a:

$$R|| \sum C_j$$

para o qual existe algoritmo que roda em tempo polinomial $O(mn^3)$ (Bruno, Coffman & Sethi - 1974).

2.6 Fluxo em redes

O problema da atribuição ótima (OAP) e por conseguinte o problema MRTA, são vistos em fluxo em redes como problemas de fluxo de custo mínimo, dos mais fundamentais problemas de fluxo em rede [ahu]. Genericamente estes problemas são descritos da seguinte maneira. Queremos determinar uma distribuição de mínimo custo de um produto através de uma rede para satisfazer demandas para certos nós supridos por outros nós. Problemas de fluxo em redes são modelados através de grafos com pesos nas arestas. Estes pesos representam as capacidades máximas das respectivas arestas.

Definição 2.6.1 (Problema de fluxo de custo mínimo) [ahu] *Formalmente temos um grafo dirigido $G = (N, E)$ definido por um conjunto de N de nós e um conjunto E de arestas dirigidas. Cada aresta $(i, j) \in E$ tem a si associado um custo c_{ij} que denota o custo por unidade de fluxo daquela aresta. Também associamos a cada aresta $(i, j) \in E$ uma capacidade u_{ij} que denota a máxima quantidade que pode fluir pela aresta e l_{ij} denota a mínima quantidade que deve fluir pelo arco. Associamos a cada nó $i \in N$ um inteiro $b(i)$ representando suprimento/demanda. Se $b(i) > 0$ o nó é uma fonte e se $b(i) < 0$ é um sorvedouro com demanda $-b(i)$. X_{ij} representa o fluxo no arco $(i, j) \in E$. Ou seja,*

Minimizar

$$\sum_{(i,j) \in A} c_{ij} x_{ij}$$

sujeito a

$$\sum_{\{j:(i,j) \in A\}} x_{ij} - \sum_{\{j:(j,i) \in A\}} x_{ji} = b(i) \quad \text{para todo } i \in N$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \quad \text{para todo } (i, j) \in A$$

onde

$$\sum_{i=1}^n b(i) = 0$$

Ou seja, encontrar o caminho mais barato para satisfazer demandas, sujeito a restrições de fonte e capacidade. Para o MRTA construímos um grafo bipartido $G = (R \cup T, E)$, com $n = |R| = |T|$ (R é o conjunto de robôs, T é o conjunto de tarefas e E é o conjunto dos possíveis pares robô-tarefa). O custo c_e associado a cada aresta é o custo estimado do par robô-tarefa. Então adicionamos a G um nó fonte p com demanda $b_p = n$ e um nó sorvedouro q com demanda $b_q = -n$. Para cada robô $r_i \in R$, adicionamos a G uma aresta de custo zero ligando p com r_i e para cada tarefa $t_i \in T$, adicionamos a G uma aresta de custo zero ligando t_i e q . As capacidades k_e para todas as arestas no grafo são 1. Resolvemos o problema encontrando o fluxo integral de custo mínimo no grafo aumentado. Para isto podemos, pex, encontrar sucessivos menores caminhos da fonte para o sorvedouro. O algoritmo de Dijkstra sobre heaps de Fibonacci resolve o problema MRTA descrito em tempo $O(mn + n^2 \log n)$.

3 Taxonomia

Em [Ger03] é apresentada uma taxonomia para sistemas multi-robóticos, que difere de taxonomias propostas anteriormente [DJMW96, CFK95], pois descreve sistemas multi-robóticos com foco nos tipos de problemas que caracterizam tais sistemas e não apenas sob o ponto de vista de sua arquitetura. A taxonomia proposta foi direcionada para posicionar no espaço de problemas resultante os diversos tipos de problemas para alocação de tarefas em sistemas multi-robóticos, bem como explicar como a teoria organizacional se relaciona com estes problemas e com as soluções propostas pela literatura robótica. Para isto são adotados três pilares que serão utilizados na descrição de problemas de alocação de tarefas em sistemas multi-robóticos:

- Single-task robots(ST) X Multi-task robots(MT):
Robôs de tarefa única vs. robôs multi-tarefa, ou seja, um robô é capaz de executar uma única tarefa ou mais de uma tarefa ao mesmo tempo?
- Single-robot tasks(SR) X Multi-robot tasks (MR):
Tarefas de robô único vs. tarefas multi-robô, ou seja, cada tarefa requer exatamente um robô para ser executada ou algumas tarefas requerem mais de um robô?
- Instantaneous assignment(IA) X Time-extended assignment(TE):
Atribuição instantânea vs. atribuição estendida no tempo, ou seja, se instantaneamente alocarmos todos os robôs para as tarefas ou se teremos mais tarefas que robôs, por exemplo.

3.1 ST-SR-IA Robôs de tarefa única, tarefa de robô único, atribuição instantânea

É o mais simples dos problemas MRTA e se pudermos distribuir as utilidades dos robôs para todas as máquinas ou coletar estas utilidades em uma só máquina, poderemos, como visto anteriormente, encontrar a solução ótima em tempo $O(n^3)$, usando o algoritmo Munkre-Kuhn. Para isto devemos modelar este problema como uma instância do problema OAP. A alternativa a esta abordagem seria o uso de um algoritmo distribuído baseado em leilões, que propositalmente não foi abordado neste documento por não fazer parte do escopo atual deste estudo (é uma alternativa melhor para a comunicação, mas pior no desempenho, ou seja, a alternativa de um algoritmo centralizado ainda é melhor). A desvantagem do uso do algoritmo centralizado é que este necessita enviar n^2 mensagens para transmitir a utilidade de cada robô para cada tarefa. Pode-se utilizar otimização para melhorar o algoritmo, mas não deixa de ser um ponto importante a ser considerado. Se não formos utilizar um sistema de grande escala de robôs (mais de 200 robôs), podemos usar o algoritmo ótimo, sem nos preocuparmos com o problema da comunicação (fazendo broadcast, pex). A solução sub-ótima do algoritmo guloso pode ser bastante interessante se o problema não apresentar combinações patológicas entre as utilidades dos robôs como é o caso de um problema típico MRTA.

3.2 ST-SR-TE Robôs de tarefa única, tarefa de robô único, atribuição estendida no tempo

Um exemplo deste caso ocorre quando temos mais tarefas que robôs e podemos prever com boa precisão futuras utilidades para as tarefas. É uma instância da classe de problemas de scheduling vista anteriormente:

$$R || \sum w_j C_j$$

Sabemos, também, que estamos diante de uma classe que é *NP*-difícil. Podemos, no entanto, recorrer a seguinte estratégia. Se temos, m robôs e n tarefas, onde $m > n$, resolvemos inicialmente o problema da atribuição ótima $m \times n$ e usamos um algoritmo guloso para atribuir as tarefas que faltam, conforme os robôs fiquem disponíveis ao longo do tempo. Este algoritmo guloso é 3-competitivo e quanto mais nos aproximarmos da alocação instantânea, mais próximos estaremos da solução ótima, ou seja, quando $(n - m) \rightarrow 0$. Uma alternativa foi apresentada por [Par98] e trata-se de uma abordagem em aprendizado, na qual os robôs aprendem empiricamente suas estimativas de utilidade e seus algoritmos de scheduling. Apresentou resultados melhores para domínios específicos.

3.3 ST-MR-IA Robôs de tarefa única, tarefas multi-robô, atribuição instantânea

Aqui para realizar uma tarefa, precisamos de um grupo de robôs. Portanto, Devemos considerar a combinação das utilidades dos grupos de robôs. Note que não trata-se simplesmente de somar as utilidades, mas de levar em consideração se alguém do grupo pode ou não executar um aspecto mandatório da tarefa. De nada vale o grupo ter a maior utilidade se ninguém puder realizar tal aspecto.

Definição 3.3.1 (Partição de um conjunto) *Uma família X é uma partição do conjunto E se e somente se os elementos de X forem mutuamente disjuntos e sua união for E .*

Definição 3.3.2 (Problema do particionamento de um conjunto (SPP)) *Dado um conjunto finito E , uma família F de subconjuntos aceitáveis de E , e uma função de utilidade $u : F \rightarrow \mathbb{R}_+$ encontre uma família de utilidade máxima X dos elementos em F tal que X seja uma partição de E .*

Se considerarmos E como o conjunto de robôs, F como o conjunto de todos os pares possíveis de grupos de robôs e tarefas e u como a estimativa de utilidade para cada tal par, teremos então modelado o problema ST-MR-IA como uma instância do SPP. O problema é que SPP é NP -difícil. Apesar disto temos boas heurísticas para este problema (exatas e aproximadas) advindas de problemas de scheduling de tripulação para aeronaves. Algumas já foram adaptadas para nosso problema e apresentaram ótimos resultados.

3.4 ST-MR-TE Robôs de tarefa única, tarefas multi-robô, atribuição estendida no tempo

Este problema envolve formação de grupo e scheduling ao mesmo tempo. Para chegarmos à solução ótima precisaremos levar em conta todas as possíveis formações distribuídas no tempo de todas as possíveis formas, ou seja, também um problema NP -difícil. Uma forma de abordar este problema é ignorar a característica do tempo estendido e aproximar para uma instância do problema ST-MR-IA iterado no tempo. Podemos, por exemplo, utilizar o seguinte algoritmo:

1. Se qualquer robô permanece disponível, encontre um par robô-tarefa de maior utilidade. Senão saia.
2. Atribua este robô para esta tarefa e o coloque como indisponível.
3. Vá ao passo 1.

Este é um algoritmo guloso 2-competitivo, ou seja, esperamos soluções sub-ótimas que no pior caso dará metade do benefício da solução ótima.

3.5 MT-SR-IA e MT-SR-TE Robôs multi-tarefa, tarefa de robô único

Ambos os problemas são incomuns, pois pressupõem que teremos robôs que executem mais de uma tarefa ao mesmo tempo. De qualquer forma, o problema MT-SR-IA pode ser tratado como resolver o problema ST-SR-IA com robôs e tarefas trocados na formulação do SPP. O mesmo pode ser dito para o problema MT-SR-TE em relação ao problema ST-SR-TE.

3.6 MT-MR-IA Robôs multi-tarefa, tarefas multi-robô, atribuição instantânea

Neste caso temos robôs multi-tarefa e tarefas multi-robô. Temos então um problema de cobertura.

Definição 3.6.1 (Cobertura) *Uma família X é uma cobertura do conjunto E se e somente se a união dos elementos de X for E .*

Os subconjuntos em uma cobertura não precisam ser disjuntos (como em partição).

Definição 3.6.2 (Problema da Cobertura em conjuntos (SCP)) *Dado um conjunto finito E , uma família F de subconjuntos aceitáveis de E , e uma função de custo $u : F \rightarrow \mathbb{R}_+$ encontre uma família de custo mínimo X de elementos em F tal que X seja uma cobertura de E .*

Se considerarmos E como o conjunto de robôs, F como o conjunto de todos os pares factíveis de grupos de robôs e tarefas (podendo ocorrer sobreposição) e c como a estimativa de custo para cada tal par, teremos então modelado o problema MT-MR-IA como uma instância do SCP. O problema é que SCP também é *NP*-difícil. Existe, no entanto, um algoritmo guloso de aproximação que tem fator competitivo logarítmico (no tamanho do maior subconjunto factível) e seu tempo de execução é polinomial no número de subconjuntos factíveis ($|F|$). Uma outra heurística apresentada por Bar-Yehuda & Even(1981) tem fator competitivo igual ao máximo número de subconjuntos para os quais qualquer elemento pertença e seu tempo de execução é a soma dos tamanhos dos subconjuntos factíveis. Estes algoritmos têm bom desempenho quando o espaço dos subconjuntos factíveis é limitado. Em MRTA é o equivalente a termos o espaço das possíveis formações de grupos de robôs naturalmente limitada. Um exemplo é quando tivermos robôs heterogêneos e/ou fisicamente separados por grande distância.

3.7 MT-MR-TE Robôs multi-tarefa, tarefas multi-robô, atribuição estendida no tempo

Este problema é *NP*-difícil e é uma instância do problema de scheduling com tarefas multiprocessadas e máquinas de múltiplo propósito.

$$MPTmMPMn|| \sum w_j C_j$$

Não é certo se existe alguma heurística ou algoritmo de aproximação para este problema.

4 Algoritmo Munkre-Khun

Será apresentada abaixo a simulação do algoritmo de Munkre-Khun, que é uma variação do algoritmo Húngaro com pesos nas arestas. Este algoritmo trata o problema da atribuição. Por exemplo, dados n trabalhadores e n tarefas, onde cada trabalhador tem um custo fixo por tarefa, determinar um emparelhamento perfeito entre trabalhadores e tarefas de custo mínimo. A seguir apresentarei o algoritmo através de uma simulação do mesmo.

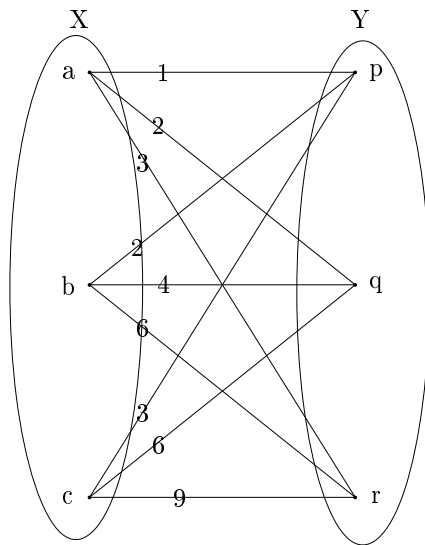
4.1 Descrição

Seja G um grafo $(X, Y) - bipartido$ completo com $|V(X)| = |V(Y)|$ e com pesos em suas arestas. A seguir os passos do algoritmo húngaro:

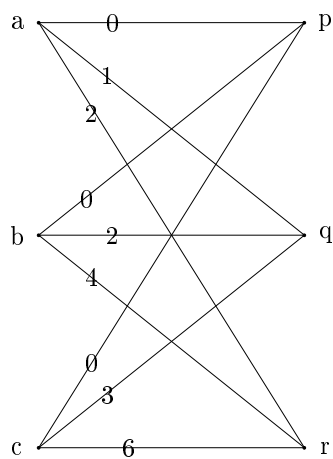
- Passo 1
Para cada vértice do subconjunto X da bipartição: Escolha a aresta incidente de menor peso e subtraia este valor de todas as arestas incidentes neste vértice. Quando tiver terminado vá ao passo 2.
- Passo 2
Coloque asterisco em cada aresta de peso 0 de G , desde que nenhuma adjacente a aresta considerada esteja marcada com 0^* . Feito isto vá ao passo 3.
- Passo 3
Para cada aresta marcada como 0^* : cubra esta aresta e todas as suas adjacentes no subconjunto Y da bipartição.
- Passo 4
Procure por uma aresta descoberta de peso 0 e faça-a $0'$ (se não houver uma tal aresta, vá ao passo 6) . Se houver alguma aresta adjacente a esta aresta que seja 0^* , descubra as adjacentes a 0^* em Y e cubra as adjacentes em X desta aresta recentemente marcada como $0'$. Se não houver nenhuma aresta adjacente a $0'$ que seja 0^* vá ao passo 5. Repita o passo 4 enquanto houver arestas de peso 0 descobertas.
- Passo 5
A partir da aresta $0'$ encontrada no passo 4, encontre um caminho E-aumentador, ou seja, deve começar por esta aresta $0'$ e terminar com outra aresta $0'$, alternando entre $0'$ e 0^* . Todas as arestas 0^* do caminho devem ser alteradas para 0 e todas as arestas do caminho que são $0'$ devem ser alteradas para 0^* . Descubra todas as arestas de G e retorne ao passo 3.
- Passo 6
Procure pela aresta descoberta de menor peso e subtraia seu peso de cada aresta descoberta (dela inclusive). Some este mesmo peso em cada aresta coberta em X . Não altere mais nada e volte ao passo 4.

4.2 Simulação

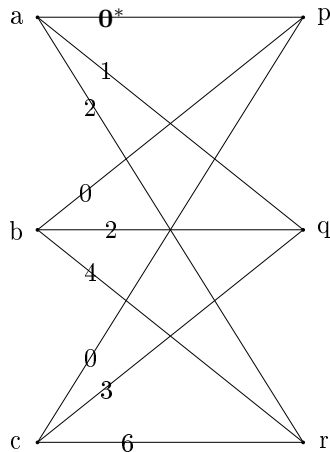
Para exemplificar vamos supor 3 trabalhadores (a,b,c) e 3 tarefas (p,q,r) com os custos dados na matriz abaixo, bem como no grafo XY-bipartido que representa esta matriz:



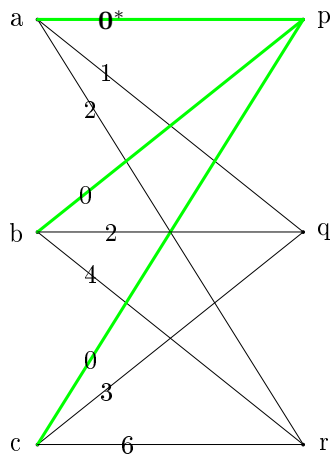
Vamos executar o passo 1 do algoritmo no grafo acima, ou seja, para cada vértice de X: escolher a aresta de menor peso incidente no vértice considerado e subtrair este peso de cada aresta incidente neste vértice. No caso as arestas escolhidas tem peso, respectivamente, 1, 2, 3.



Agora vamos ao passo 2 do algoritmo, marcando com 0^* cada aresta que possua peso 0 e não possua aresta adjacente marcada como 0^* , ou seja, em nosso exemplo apenas a aresta ap é escolhida:

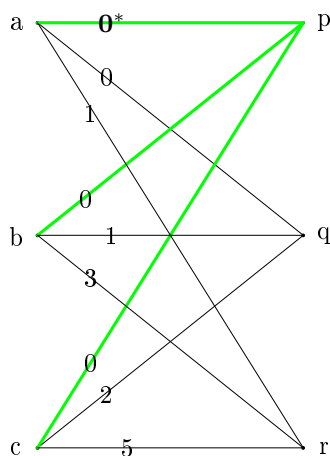


Passamos, então, ao passo 3 e vamos cobrir cada aresta adjacente a arestas do tipo 0^* (adjacentes no subconjunto Y da bipartição).

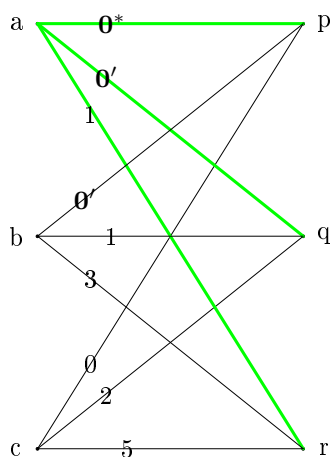


O passo 4 recomenda que encontremos uma aresta de peso 0 e descoberta. Como não existe tal aresta vamos direto para o passo 6. No passo 6 devemos encontrar uma aresta descoberta de menor peso e subtrair seu peso de todas as arestas descobertas. Devemos também somar seu peso com arestas que foram cobertas por serem adjacentes a uma 0^* no subconjunto X da bipartição. Tam-

bém não devemos alterar nenhuma cobertura. No nosso caso a aresta de menor peso é a aresta aq de peso 1. Após todo o processo teremos:

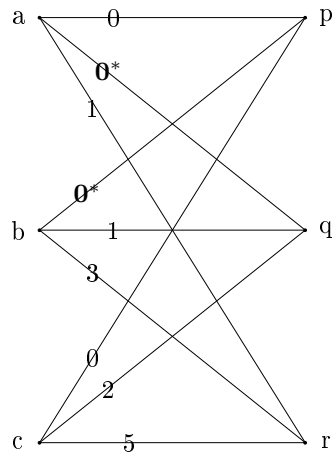


De volta ao passo 4 devemos procurar por uma aresta de peso 0 que não esteja coberta e fazê-la $0'$. No nosso caso esta aresta é adjacente a uma aresta 0^* através do subconjunto X da bipartição, ou seja, devemos descobrir as adjacentes a 0^* através do subconjunto Y da bipartição e cobrir as adjacentes a $0'$ através do subconjunto X da bipartição. Ainda no passo 4 devemos averiguar se ainda existe aresta de peso 0 não coberta. No nosso caso tal aresta é a aresta bp . Então fazemos esta aresta $0'$. Como ela não possui adjacentes 0^* no subconjunto X da bipartição, vamos ao passo 5. Após o processo todo do passo 4 temos:

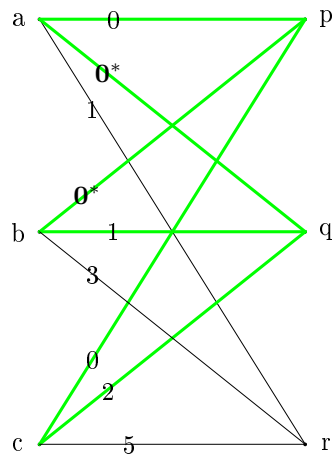


No passo 5 vamos encontrar uma caminho E-aumentador que comece na aresta $0'$ e termine em outra $0'$, alternando $0'$ com 0^* . No nosso caso o caminho é $bp - pa - aq$. Feito isto devemos fazer as aresta 0^* em 0 e as arestas $0'$ em

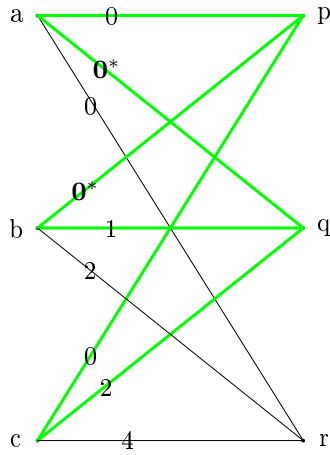
0^* . Retiramos todas as coberturas e vamos ao passo 3. Após todo o processo temos:



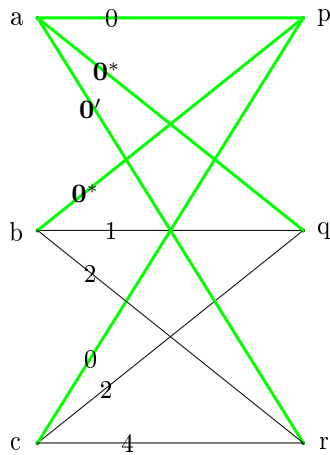
O passo 3 nos diz para cobrir todas as adjacentes a arestas 0^* através do subconjunto X da bipartição. Teremos:



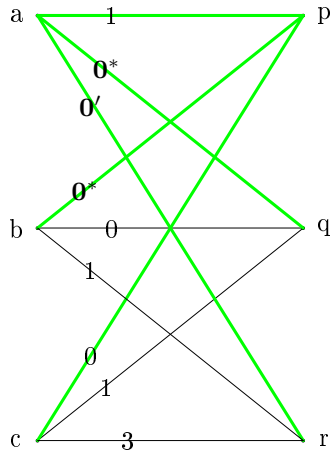
Vamos ao passo 4. Devemos procurar por arestas de peso 0 que não estejam cobertas. Como não conseguimos encontrar nenhuma aresta deste tipo, devemos ir ao passo 6. No passo 6 encontramos a aresta descoberta de menor peso e efetuamos as subtrações e adições (se necessárias) conforme descrito anteriormente. No nosso caso a aresta de menor peso é a aresta ar de peso 1:



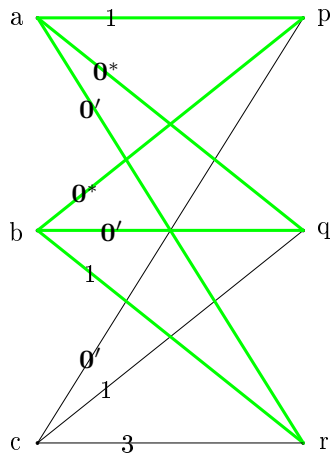
No passo 4 procuramos por aresta de peso 0 que não seja coberta. A aresta ar é encontrada e como possui adjacente em X do tipo 0^* , invertemos as coberturas conforme descrito anteriormente e procuramos novamente por arestas de peso 0 descobertas. Como não encontramos, vamos ao passo 6. Após o passo 4 temos:



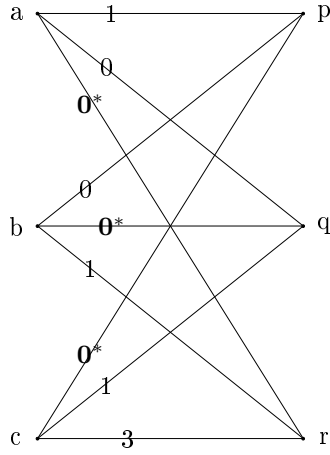
Executamos o passo 6 conforme descrito anteriormente e encontraremos a aresta bq de peso 1. Vale observar que desta não teremos apenas subtrações, mas também teremos uma adição na aresta ap . Após o processo todo teremos:



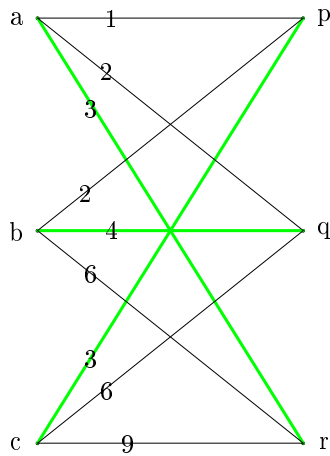
No passo 4 teremos a aresta bq transformada em aresta $0'$, ocorrendo a inversão de coberturas. Feito isto voltamos a procurar por arestas descobertas de peso 0 e encontraremos a aresta cp , que por não possuir adjacente em X do tipo 0^* , nos levará ao passo 5. Após este passo 4 temos:



O caminho E-aumentador encontrado no passo 5 é formado pelas arestas $cp - pb - bq - qa - ar$. Após todo este passo teremos:



E de volta ao passo 3 teremos todas as arestas do grafo cobertas, terminamos. Logo o emparelhamento procurado é:



5 Conclusão e Futuro

Através do presente trabalho iniciei a pesquisa em alocação de tarefas para sistemas multi-robóticos (MRTA) usando como base a pesquisa existente em diversas áreas para o problema da atribuição ótima (OAP). Para isto utilizei a recente tese de doutorado de Brian Gerkey [Ger03] (e seus desdobramentos bibliográficos). Em sua tese, Gerkey criou uma taxonomia para a área com bases formais e a idéia era iniciar com base sólida a pesquisa que se seguirá, ou seja, a análise de uma métrica muito pouco utilizada como função utilidade nos robôs (um trabalho que fez uso de tal métrica foi de [SB99]). Esta métrica faz uso do plano do robô como estimativa de sua futura performance. Para isto terei de levar em conta aspectos quantitativos e qualitativos do plano. Inicialmente

levarei em conta apenas o tamanho do plano (quantitativo). Depois pesquisarei a sintaxe do plano (qualitativo). Outro ponto importante a ser ressaltado sobre o presente trabalho é que dentre os diversos algoritmos e as diversas heurísticas citadas na tese de Gerkey escolhi o algoritmo de Munkre-Kuhn para um estudo mais detalhado, com simulação e implementação, pois a pesquisa futura focará um sistema do tipo ST-SR-IA.

6 Apêndice

```

/*****
Algoritmo                                                                 Húngaro
*****/
Autor: Antonio Luiz Basile IME-USP-2004 Orientador: Prof. Flávio Soares
Corrêa da Silva

Um grafo com as arestas discriminadas em subX e subY (emparelhamento
perfeito) será criado e os pesos destas arestas serão gerados
aleatoriamente. O predicado aresta (e aresta2) é descrito conforme:
aresta(Vertice X, Vertice Y, Peso, Condição, Cobertura) onde
Condição pode ser "none" ou "star" ou "primed" e Cobertura pode
ser "clear" ou "coverX" ou "coverY". Inicialmente todas as arestas
do grafo começam com "none".

A pesquisa é hung.
*****/

:- dynamic(aresta/6). %usado nos cálculos
:- dynamic(aresta2/6). %usado para guardar pesos iniciais
:- dynamic(subX/1).

hung :- cria, passo1, passo2, !, passo3. %ponto inicial do programa

/*****
cria:
*****/
Gera o grafo com pesos aleatórios nas arestas. É um emparelhamento
perfeito e os subconjuntos X e Y da bipartição são definidos pelos
predicados subX e subY, respectivamente.
*****/

cria :-
    retractall(aresta(_,_,_,_,_)),
    retractall(aresta2(_,_,_,_,_)),
    subX(Lx),
    subY(Ly),
    criador1(Lx, Ly).

criador1([ ], _).
criador1([X|Xs], L) :-

```



```

        criador2(X,L),
        criador1(Xs,L).

criador2(_,[_]).
criador2(X,[Y|Ys]) :-
    A is random(50),
    B is A+1,
    assertz(aresta(X,Y,B,none,clear,clear)),
    assertz(aresta2(X,Y,B,none,clear,clear)),
    criador2(X,Ys).

subX([xa,xb,xc,xd,xe,xf,xg,xh,xi,xj,xk,xl,xm,xn]).
subY([ya,yb,yc,yd,ye,yf,yg,yh,yi,yj,yk,yl,ym,yn]).

/*****
passo1 :
*****/
Para cada vértice do subconjunto X da bipartição, escolhe as
arestas de menor peso em relação as suas adjacentes (no
subconjunto X da bipartição). Feito isso subtrai este peso de
todas as incidentes naquele vértice.
*****/

passo1 :-
    subX(X),
    achaMenores(X,R),
    subtrai(R),
    writeln('passo1->subtraindo menor peso de adjacentes em X'),
    writeln('').

subtrai([_]).
subtrai([(A,B)|As]) :-
    aresta(A,B,Peso,Cond,Cobx,Coby),
    findall(X,aresta(A,X,_,_,_,_),L),
    subtrai(A,L,Peso),
    subtrai(As).

subtrai(A,[_],P).
subtrai(A,[B|Bs],P) :-
    aresta(A,B,Peso,Cond,Cobx,Coby),
    T is Peso-P,
    retract(aresta(A,B,Peso,Cond,Cobx,Coby)),
    assertz(aresta(A,B,T,Cond,Cobx,Coby)),
    subtrai(A,Bs,P).

achaMenores(L,R) :- achaMenores(L,[_],R).

achaMenores([_],A,R) :- reverse(A,R).

achaMenores([A|As], S, R) :-

```



```

        findall((A,Vy,Peso),
                aresta(A,Vy,Peso,Cond,Cobx,Coby),
                L),
        achaMenor(L, Vy),
        achaMenores(As, [(A,Vy)|S], R).

achaMenor([(A,Vy,Peso)|Resto], Resp) :-
    achaMenor(Resto, Vy, Peso, Resp).

achaMenor([ ], Resp, Peso, Resp).

achaMenor([(A,Vy,Peso)|Resto], Y, Py, Resp) :-
    Peso < Py,
    achaMenor(Resto, Vy, Peso, Resp).

achaMenor([(A,Vy,Peso)|Resto], Y, Py, Resp) :-
    Peso >= Py,
    achaMenor(Resto, Y, Py, Resp).

/*****
passo2 :
*****/
Transforma cada aresta de peso 0 em 0*, desde que ela não possua
adjacente 0*.
*****/

passo2 :-
    findall((A,B),
            aresta(A,B,0,Cond,Cobx,Coby),
            L),
    sort(L,L2),
    passo2(L2).

passo2([ ]) :-
    writeln('passo2->arestas peso 0 viram 0* se não possuem adj 0*'),
    writeln('').

passo2([(A,B)|Resto]) :-
    findall((X,B),
            (aresta(X,B,0,star,Cobx,Coby), X\=A),
            L2),
    findall((A,Y),
            (aresta(A,Y,0,star,Cobx,Coby), Y\=B),
            L3),
    append(L2,L3,L4),
    length(L4,N),
    gravaPasso2(A,B,N),
    passo2(Resto).

gravaPasso2(A,B,N) :- N \= 0.

```



```

gravaPasso2(A,B,0) :-
    aresta(A,B,0,Cond,Cobx,Coby),
    retract(aresta(A,B,0,_,Cobx,Coby)),
    asserta(aresta(A,B,0,star,Cobx,Coby)).

/*****
passo3 :
*****/
Cobre todas as arestas 0* e suas adjacentes através do subconjunto
Y da bipartição.
*****/

passo3 :-
    findall((A,B),
        (aresta(A,B,P,Cond,Cx,Cy),P=0,Cond=star,Cy=clear),
        L),
    passo3(L).

passo3([ ]) :-
    verificafim,
    writeln(''),
    write('0 emparelhamento indicado é ---> '),
    findall((A,B),
        aresta(A,B,Peso,star,Cobx,Coby),
        L),
    sort(L,L2),
    writeln(L2),
    pegapeso(L,L3),
    sort(L3,L4),
    mostralista(L4),
    somalista(L4,R4),
    write('0 peso total é '),
    writeln(R4),
    writeln('').

passo3([ ]) :- passo4.

passo3([(A,B)|Resto]) :-
    writeln('passo3->cobrindo arestas 0* e adjacentes em Y'),
    writeln(''),
    findall((X,B),
        (aresta(X,B,Peso,Cond,Cobx,Coby),Coby=clear),
        L),
    cobrearesta(L),
    passo3(Resto).

pegapeso([ ],[ ]).
pegapeso([(A,B)|Resto],[ (A,B,Peso)|R]) :-
    aresta2(A,B,Peso,_,_,_),

```



```

    pegapeso(Resto,R).

mostralista([ ]).
mostralista([A|B]) :-
    writeln(A),
    mostralista(B).

somalista([ ],0).
somalista([(X,Y,Peso)|B],R) :-
    somalista(B,T),
    R is T+Peso.

verificafim :-
    findall(A,aresta(A,B,P,star,Cobx,Coby),L1),
    list_to_set(L1,L3),
    subX(L2),
    length(L2,M),!,
    length(L3,M).

cobrearesta([ ]).
cobrearesta([(X,B)|Resto]) :-
    aresta(X,B,Peso,Cond,Cobx,Coby),
    retract(aresta(X,B,Peso,Cond,Cobx,Coby)),
    assert(aresta(X,B,Peso,Cond,Cobx,B)),
    cobrearesta(Resto).

/*****
passo4 :
*****/
Procura poruma aresta descoberta de peso 0 e a faz 0' (se não há
tal aresta, vai ao passo 6) . Se houver aresta adjacente a esta
aresta que é 0* (em X), descobre as adjacentes a 0* em Y e cobre
as adjacentes em X desta aresta recentemente marcada como 0'. Se
não há nenhuma aresta adjacente a 0' que é 0*, vai para o passo 5.
Repita o passo 4 enquanto houver arestas de peso 0 descobertas.
*****/

passo4 :-
    findall((A,B),
        (aresta(A,B,0,Cond,Cx,Cy),Cx=clear,Cy=clear),
        L),
    sort(L,L1),
    passo4(L1).

passo4([ ]) :-
    writeln('passo4indo6->no passo6 vou criar novos zeros'),
    writeln(''),
    !,
    passo6.

```



```

passo4([(A,B)|Resto]) :-
    retract(aresta(A,B,0,Cond,Cobx,Coby)),
    assert(aresta(A,B,0,primed,Cobx,Coby)),!,
    passo4(A,B).

passo4(A,B) :-
    writeln('passo4indo4->criando 0prim e invertendo cobs'),
    writeln(''),
    aresta(A,X,Peso,star,Cobx,Coby),
    inverteCob((A,B),(A,X)),
    !,
    passo4.

passo4(A,B) :-
    writeln('passo4indo5->procurar caminho E-aum no passo5'),
    writeln(''),
    !,
    passo5(A,B).

inverteCob((A,B),(A,X)) :-
    findall((M,X),
        aresta(M,X,Peso,Cond,Cobx,Coby),
        Lstar),
    findall((A,N),
        aresta(A,N,Peso,Cond,Cobx,Coby),
        Lprim),
    retiraCob(Lstar),
    cobreX(Lprim).

retiraCob([ ]).
retiraCob([(A,B)|Resto]) :-
    aresta(A,B,Peso,Cond,Cobx,Coby),
    retract(aresta(A,B,Peso,Cond,Cobx,Coby)),
    asserta(aresta(A,B,Peso,Cond,Cobx,clear)),
    retiraCob(Resto).

cobreX([ ]).
cobreX([(A,B)|Resto]) :-
    aresta(A,B,Peso,Cond,Cobx,Coby),
    retract(aresta(A,B,Peso,Cond,Cobx,Coby)),
    asserta(aresta(A,B,Peso,Cond,A,Coby)),
    cobreX(Resto).

/*****
passo5 :
*****/
A partir da aresta 0' encontrada no passo 4, encontre um caminho
E-aumentador, ou seja, deve começar por esta aresta 0' e terminar
com outra aresta 0', alternando entre 0' e 0*. Todas as arestas 0*
do caminho devem ser alteradas para 0 e todas as arestas do caminho

```


que são 0' devem ser alteradas para 0*. Descubra todas as arestas de G e retorne ao passo 3.

*****/

```
passo5(A,B) :-
    aresta(A, B, PesoAB, primed, CobxAB, CobyAB),
    aresta(M, B, PesoMB, star, CobxMB, CobyMB),
    aresta(M, C, PesoMC, primed, CobxMC, CobyMC),
    append(L, [(A,B), (M,B), (M,C)], L2),
    passo5a(L2, (M,C)).
```

```
passo5(A,B) :- !, passo5b([ ], (A,B)).
```

```
passo5a(L, (A,B)) :-
    aresta(A, B, PesoAB, primed, CobxAB, CobyAB),
    aresta(M, B, PesoMB, star, CobxMB, CobyMB),
    aresta(M, C, PesoMC, primed, CobxMC, CobyMC),
    append(L, [(A,B), (M,B), (M,C)], L2),
    passo5a(L2, (M,C)).
```

```
passo5a(L, (A,B)) :-
    tiracob,
    sort(L, L2),
    mudaStarPrimed(L2),
    write('passo5->o caminho E-aumentador encontrado é '),
    writeln(L2),
    writeln('vou voltar para o passo 3 e cobrir os novos 0*'),
    writeln(''),
    passo3.
```

```
passo5b(L, (A,B)) :-
    aresta(A, B, PesoAB, primed, CobxAB, CobyAB),
    not(aresta(M, B, PesoMB, star, CobxMB, CobyMB)),
    retract(aresta(A, B, PesoAB, primed, CobxAB, CobyAB)),
    asserta(aresta(A, B, PesoAB, star, CobxAB, CobyAB)),
    tiracob,
    passo3.
```

```
passo5b(L, (A,B)) :-
    aresta(A, B, PesoAB, primed, CobxAB, CobyAB),
    aresta(M, B, PesoMB, star, CobxMB, CobyMB),
    not(aresta(M, C, PesoMC, primed, CobxMC, CobyMC)),
    retract(aresta(A, B, PesoAB, primed, CobxAB, CobyAB)),
    asserta(aresta(A, B, PesoAB, star, CobxAB, CobyAB)),
    tiracob,
    passo3.
```

```
tiracob :-
    findall((A,B),
        aresta(A, B, P, Cond, Cobx, Coby),
```



```

        L),
    retiraCob2(L).

retiraCob2([ ]).
retiraCob2([(A,B)|Resto]) :-
    aresta(A,B,Peso,Cond,Cobx,Coby),
    retract(aresta(A,B,Peso,Cond,Cobx,Coby)),
    asserta(aresta(A,B,Peso,Cond,clear,clear)),
    retiraCob2(Resto).

mudaStarPrimed([ ]).
mudaStarPrimed([(A,B)|Resto]) :-
    aresta(A,B,P,star,Cobx,Coby),
    retract(aresta(A,B,P,star,Cobx,Coby)),
    asserta(aresta(A,B,P,none,Cobx,Coby)),
    mudaStarPrimed(Resto).
mudaStarPrimed([(A,B)|Resto]) :-
    aresta(A,B,P,primed,Cobx,Coby),
    retract(aresta(A,B,P,primed,Cobx,Coby)),
    asserta(aresta(A,B,P,star,Cobx,Coby)),
    mudaStarPrimed(Resto).

/*****
passo6 :
*****/
Procura pela aresta descoberta de menor peso e subtrai seu peso de
cada aresta descoberta (dela inclusive). Soma este mesmo peso em
cada aresta coberta em X. Não altera mais nada e volte ao passo 4.
*****/

passo6 :-
    findall((A,B,PesoAB),
        aresta(A,B,PesoAB,CondAB,clear,clear),
        L1),
    findall((M,N,PesoMN),
        (aresta(M,N,PesoMN,none,CX,CY),CX\=clear),
        L2),
    achaMenorLivre(L1, Pmen, (X,Y)),
    subtraiLivre(L1,(X,Y,Pmen)),
    somaCobX(L2,(X,Y,Pmen)),
    writeln('passo6->procurado pela aresta descob de menor peso'),
    writeln(''),
    passo4.

somaCobX([ ],_).
somaCobX([(A,B,Peso)|As],(X,Y,Pmen)) :-
    aresta(A,B,Peso,Cond,Cobx,Coby),
    T is Peso+Pmen,
    retract(aresta(A,B,Peso,Cond,Cobx,Coby)),
    assertz(aresta(A,B,T,Cond,Cobx,Coby)),

```



```

somaCobX(As, (X, Y, Pmen)).

subtrailivre([ ], _).
subtrailivre([(A, B, Peso) | As], (X, Y, Pmen)) :-
    aresta(A, B, Peso, Cond, Cobx, Coby),
    T is Peso - Pmen,
    retract(aresta(A, B, Peso, Cond, Cobx, Coby)),
    assertz(aresta(A, B, T, Cond, Cobx, Coby)),
    subtrailivre(As, (X, Y, Pmen)).

achaMenorLivre([(A, B, Peso) | Resto], Pmen, Resp) :-
    achaMenorLivre(Resto, (A, B), Peso, Pmen, Resp).

achaMenorLivre([ ], Resp, Peso, Peso, Resp).

achaMenorLivre([(A, B, Peso) | Resto], (Va, Vb), Pab, Pmen, Resp) :-
    Peso < Pab,
    achaMenorLivre(Resto, (A, B), Peso, Pmen, Resp).

achaMenorLivre([(A, B, Peso) | Resto], (Va, Vb), Pab, Pmen, Resp) :-
    Peso >= Pab,
    achaMenorLivre(Resto, (Va, Vb), Pab, Pmen, Resp).

```

Referências

- [ahu] *Network Flows - Theory, Algorithms and Applications*, chapter 01.
- [bon77] *Graph Theory with Applications*, chapter 05. 1977.
- [Bot98] Silvia Botelho. A distributed scheme for task planning and negotiation in multi-robot systems, 1998.
- [CFK95] Yu Uny Cao, Alex S. Fukunaga, and Andrew B. Kahng. Cooperative mobile robotics: Antecedents and directions. Technical Report 950049, 1995.
- [DJMW96] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. A taxonomy for multi-agent robotics, 1996.
- [Ger03] Brian P. Gerkey. *On Multi-Robot Task Allocation*. Phd dissertation, University of Southern California Computer Science Department, August 2003. Also Technical Report CRES-03-012.
- [GL99] G. De Giacomo and H. Levesque. An incremental interpreter for high-level programs with sensing, 1999.
- [GLLS02] Guiseppe De Giacomo, Yves Lespérance, Hector J. Levesque, and Sebastian Sardiña. On the semantics of deliberation in indigolog - from theory to implementation (extended abstract), 2002.

- [GM00] Brian P. Gerkey and Maja J. Matarić. Murdoch: Publish/subscribe task allocation for heterogeneous agents. In *International Conference on Autonomous Agents*, pages 203–204, Barcelona, Spain, Jun 2000. (Also appears as a Student Abstract in Proc. of the National Conf. on Artificial Intelligence (AAAI), Austin, Texas, Aug 2000).
- [GM01] Brian P. Gerkey and Maja J. Matarić. Principled communication for dynamic multi-robot task allocation. In D. Rus and S. Singh, editors, *Experimental Robotics VII, LNCIS 271*, pages 253–362. Springer-Verlag, Berlin, 2001. (Also appears in Proc. of the Intl. Symp. on Experimental Robotics (ISER), Waikiki, Hawaii, Dec 2000).
- [GM02a] Brian P. Gerkey and Maja J. Matarić. Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination. In *IEEE International Conference on Robotics and Automation*, pages 464–469, Washington, DC, May 2002. (Also Technical Report IRIS-01-403).
- [GM02b] Brian P. Gerkey and Maja J. Matarić. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation, Special Issue on Multi-Robot Systems*, 18(5):758–768, Oct 2002. (Also Technical Report IRIS-01-399).
- [GM03] Brian P. Gerkey and Maja J. Matarić. Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, Sep 2003. (Also Technical Report CRES-02-005).
- [GVS⁺01] Brian P. Gerkey, Richard T. Vaughan, Kasper Stoy, Andrew Howard, Gaurav Sukhatme, and Maja J. Matarić. Most valuable player: A robot device server for distributed control. In *IEEE/RSJ International Conference on Intelligent Robotics and Systems*, pages 1226 – 1231, 2001.
- [INS01] Luca Iocchi, Daniele Nardi, and Massimiliano Salerno. Reactivity and deliberation: A survey on multi-robot systems. *Lecture Notes in Computer Science*, 2103:9–??, 2001.
- [pap82] *Combinatorial Optimization, Algorithms and Complexity*, chapter 12. 1982.
- [Par98] L. Parker. Alliance: An architecture for fault-tolerant multi-robot cooperation, 1998.
- [Par00] Lynne E Parker. Current state of the art in distributed autonomous mobile robotics. In George Bekey Lynne E Parker and Jacob Barhen, editors, *Distributed Autonomous Robotic System 4*, pages 3–12. Springer-Verlag, Tokyo, October 2000. ISBN: 4-431-70295-4.
- [San93] T. W. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 295–308, Hidden Valley, Pennsylvania, 1993.

- [San98] T. Sandholm. Contract types for satisficing task allocation: I theoretical results, 1998.
- [San00] Tuomas Sandholm. Issues in Computational Vickrey Auctions. *International Journal of Electronic Commerce*, 4(3):107–129, 2000.
- [SB99] R. Alami Silvia Botelho. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement, 1999.
- [Smi80] R. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, 1980.
- [Zlo04] Robert Zlot. Complex task allocation for multirobot coordination. A thesis proposal, March 2004.