Capítulo 3

IA Multiagente: Mais Inteligência, Mais Desafios

Ana Lúcia C. Bazzan
Instituto de Informática – UFRGS
bazzan@inf.ufrgs.br

Resumo

Desde seus primórdios, a inteligência artificial se ocupa (ainda que não explicitamente) com o que hoje denominamos agente inteligente. Entretanto, esta inteligência artificial – que podemos chamar de inteligência artificial monoagente – hoje encontra o desafio de ter que lidar não apenas com um agente, mas sim com uma sociedade de agentes. Nesta, agentes munidos de intenções (possivelmente conflitantes) formam uma complexa rede de interações, isto é, um espaço social. Dado este fato, coloca-se a seguinte questão: "seria a inteligência artificial, tal como concebida entre os anos 60 e 80, adequada para tratar os desafios deste espaço social marcado pela Internet e pelos sistemas distribuídos e colaborativos" ? Neste capítulo pretende-se mostrar que a resposta é sim mas que para tal é necessário enfatizar a visão ou faceta multiagente da inteligência artificial.

Abstract

Artificial intelligence has always dealt with what is known today as an intelligent agent, even if not explicitly. However, this kind of artificial intelligence – which can be called monoagent artificial intelligence – now has the challenge of dealing with not only one agent but with a society of agents, which have (possibly conflicting) intentions and form a complex social network. Against this background, the following question can be posed: "is AI, as we know it from the 1960, adequate to face the challenges of this social space"? In this course we intend to show that the answer is yes, but it is necessary to emphasize the multiagent facet of AI.

3.1. Introdução

Uma das metas da Inteligência Artificial (IA) tem sido a de fazer computadores simularem o raciocínio humano. Entretanto os espaços sociais nos quais os seres humanos (e suas réplicas imperfeitas) agem, tem sido praticamente relegado a um segundo plano. Com os recentes avanços em tecnologia da informação e comunicação, torna-se necessário a introdução de um novo paradigma que considere interações sociais entre os agentes inteligentes.

Este paradigma começa a ser criado com o surgimento de sistemas multiagentes. Esta área não estuda nem sistemas físicos nem agentes isolados mas sim um agente como parte de um espaço social. Enquanto a IA tem como foco a investigação de porções de informação, sem necessariamente considerar como elas interagem, um sistema multiagente tenta investigar informação como um fenômeno social, o que tornou-se importante dado o atual contexto onde unidades embarcadas de processamento são ubíquas.

Em suma, a imagem de uma única mente inteligente, um oráculo que responde perguntas, ou uma máquina passiva que almeje passar no teste de Turing, hoje dá lugar a agentes ativos, atores sociais participativos e potencialmente colaborativos (como por exemplo a wikipedia), formando um espaço social de interações.

Conforme colocado por Wooldridge no apêndice de [Wooldridge 2002], a ideia de agência e agentes, já presente nos primórdios da IA, tendia originalmente à questão de um sistema *integrado* capaz de realizar ações autônomas. Entretanto, esta ideia não se realizou desta forma durante o desenvolvimento da IA. Cabe aqui analisar por que.

Desde seus primórdios, a IA teve que se ocupar com críticas advindas dos mais diversos céticos. Podemos resumir estas críticas sob a forma "computadores nunca serão capazes de X" onde X podia ser comunicar-se em linguagem natural, aprender, etc. Naturalmente a comunidade de IA passou a responder a tais críticas através da construção de sistemas que de fato pudessem, isoladamente, comunicar-se em linguagem natural ou aprender, etc., o que levou a uma "especialização" dos praticantes de IA e à criação de subáreas como processamento de linguagem natural, aprendizagem de máquina, etc.

Infelizmente houve pouco movimento no sentido de *integrar* estes componentes em sistemas holísticos. A área de agentes surgiu justamente como tentativa de integrar os diversos componentes em uma entidade denominada agente. Posteriormente, devido à interação natural entre os agentes, surge a área de sistemas multiagentes, foco deste capítulo e tópico que será discutido detalhadamente adiante.

Doravante denominaremos a IA tradicional como IA monoagente. Como será visto adiante, esta IA não apenas difere significativamente da IA multiagente que será apresentada neste capítulo, como também dificilmente atende os requisitos colocados pelo novo paradigma criado com a Internet 2.0 e os sistemas colaborativos, de natureza essencialmente social.

Por IA multiagente entende-se o uso das técnicas básicas (como por exemplo representação de conhecimento, resolução de problemas, tomada de decisão e aprendizado) em ambientes onde não apenas *um* agente age, mas sim onde vários agentes interagem bem como têm conhecimento parcial sobre o problema, o que coloca em xeque a eficiência de uma grande parte das técnicas clássicas.

Aqui deve-se observar que IA multiagente difere do que outrora se chamou

IA distribuída (IAD). Esta última (discutida na Seção 3.7), por ser essencialmente identificada, hoje, com resolução distribuída de problemas, e portanto se fincar em uma série de hipóteses muito fortes (por exemplo que o projetista do problema é o responsável pela sua decomposição em sub-tarefas), deixou de ser interessante em ambientes de natureza social complexa.

Russel e Norvig [Russell and Norvig 2004] (versão brasileira) fazem, no livro-texto que é uma referência na área de IA, um apanhado da história da IA desde a década de 60 até o presente mas se limitam a registrar o "surgimento de agentes inteligentes". No texto, os autores citam estes agentes por diversas vezes mas apenas para denominar ou ilustrar problemas monoagente clássicos, não tocando no cerne da questão que é "o que acontece quando mais de um agente está presente?".

Desta forma, o objetivo deste capítulo é: identificar os problemas da IA clássica quando aplicada a cenários multiagente, descrever o tratamento dado aos problemas acima mencionados e discorrer sobre os desafios e problemas em aberto.

Este capítulo se encontra estruturado em três partes. A parte I (Seções 3.2 a 3.6) trata de IA monoagente. Inicia-se (próxima seção) com uma revisão breve sobre IA, com foco nos tópicos mencionados anteriormente. A Seção 3.3 trata de representação de conhecimento; a Seção 3.4 aborda formalização e resolução de problemas através do formalismo de satisfação de restrições; a Seção 3.5 introduz planejamento clássico e não clássico, focando em processo de decisão de Markov para planejamento e tomada de decisão em ambientes estocásticos; a Seção 3.6 trata de aprendizagem por reforço para o caso de um agente.

A parte II traz conceitos sobre agentes autônomos e sistemas multiagente, os quais são necessários para o entendimento da parte III. As principais seções tratam de definições, terminologia, histórico e taxonomia (Seções 3.7, 3.8 e 3.9), e de coordenação e cooperação em sistemas multiagentes (Seção 3.10).

A parte III trata da IA multiagente e compreende quatro seções principais sobre lógicas para sistemas multiagentes (Seção 3.13), resolução de problemas em sistemas multiagentes (Seção 3.14), planejamento multiagente (Seção 3.15) e tomada de decisão e aprendizado em ambientes com mais de um agente (Seção 3.16).

Parte I: Inteligência Artificial Monoagente

3.2. Conceitos básicos sobre inteligência artificial

Existem diversas definições de IA. Uma lista atual pode ser encontrada no capítulo 1 de [Russell and Norvig 2004]. Para os propósitos deste capítulo, a

definição de Kurzweil é um bom ponto de partida: "inteligência artificial é a arte de criar máquinas que executam funções que exigem inteligência quando executadas por pessoas". Adiante veremos que as mesmas propriedades que Kurzweil deseja para máquinas inteligentes se aplicam a agentes inteligentes.

A IA divide-se em uma série de sub-áreas – aprendizado, representação de conhecimento, etc. – e este fato é justamente um dos pontos fracos dela, conforme dito na Seção 3.1.

Consultando textos introdutórios de IA nota-se que dois temas dominam suas partes básicas: representação de conhecimento e resolução de problemas. Não é diferente quando estendemos a IA monoagente para a IA multiagente. A resolução de problemas envolve, em ambos os casos, diversas técnicas. Num âmbito mais restrito, resolução de problemas pode ser vista como a tarefa de encontrar uma sequência de ações que leva a estados desejáveis, o que nos remete a problemas de busca, satisfação de restrições e planejamento. Num âmbito mais geral, se estamos preocupados também com a eficiência da resolução de problemas, aprendizado pode ser importante.

A seguir nos deteremos nestas sub-áreas da IA.

3.3. Representação de Conhecimento

O conceito de "agente baseado em conhecimento" está se tornando cada vez mais importante. Já na primeira edição do livro [Russel and Norvig 1995], diversos capítulos são dedicados a este tema. Na realidade, conhecimento (e sua representação) permeia a pesquisa em IA desde seus primórdios pois conhecimento e raciocínio são, não apenas fundamentais no projeto de um artefato inteligente (e.g. um agente), como também não são problemas triviais. Desta forma, projeto de bases de conhecimento (BC), entendendo-se aqui formas de representação e inferência, é uma área ativa de pesquisa.

É notório que a representação do conhecimento em problemas de IA se dá principalmente através do uso de lógica de primeira ordem. Este tipo de representação é a base da resolução de diversos problemas de IA como planejamento e diagnóstico. Entretanto há uma diferença chave entre as versões centralizada e distribuída destes problemas. No primeiro caso assume-se que o conhecimento do projetista do sistema é suficiente para encontrar um plano ou chegar a um diagnóstico. Por exemplo, a lógica de primeira ordem é suficiente para se representar o conhecimento necessário para realizar diagnóstico de componentes em circuitos lógicos como o da Figura 3.1. No segundo caso não se pode assumir que um agente conheça todas as variáveis e/ou seus valores. Na Seção 3.13 serão mostrados os problemas decorrentes do uso de lógica de primeira ordem e as implicações disto. De fato, em sistemas multiagentes, lógicas modais de conhecimento são ferramentas bastante utilizadas.

Uma BC baseada em lógica é constituída por sentenças[1] que têm uma

[1] Seguindo a terminologia utilizada em [Russell and Norvig 2004], será usado o termo "sentença" para denotar uma asserção sobre o mundo; em particular, em lógica de primeira ordem tais asserções são feitas utilizando-se uma fórmula (bem-formada).

determinada sintaxe, ou seja, são expressões corretas de uma linguagem de representação. Tal sintaxe depende do tipo particular de lógica empregada. Nesta seção nos deteremos na lógica de primeira ordem. Na Seção 3.13 veremos que a lógica modal se baseia nesta mas tem uma sintaxe estendida.

3.3.1. Sintaxe

A sintaxe de uma linguagem lógica é um sistema formal de regras que especifica como as expressões válidas da linguagem podem ser formadas a partir de um vocabulário básico.

Formalmente, uma linguagem lógica de primeira ordem é determinada pelos seguintes conjuntos de símbolos: $\mathbf{P}$ de predicados, $\mathbf{F}$ de funções, $\mathbf{C}$ de constantes e $\mathbf{V}$ de variáveis. Estes conjuntos formam o alfabeto da linguagem.

Seguindo-se a sintaxe desta linguagem são geradas fórmulas bem-formadas (ou *well-formed formula* cuja abreviação é wff) como $\neg\phi$, $((\phi(a,b) \wedge \phi(b,c)) \rightarrow \psi(a,c))$.

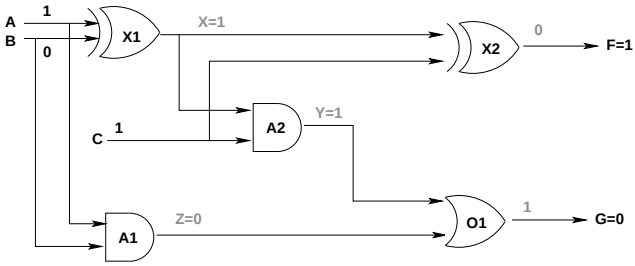

Figura 3.1. Circuito booleano

Desta forma, é possível representar o conhecimento associado ao circuito ilustrado na Figura 3.1 através de fórmulas como $Tipo(A_1, AND)$, $Saida(X_1, X)$, $Igual(X, 1)$ ou $Conectados(Saida(X_1), Entrada(1, X_2))$.

3.3.2. Semântica

Além de regras sintáticas, também é preciso se definir o *significado* das fórmulas lógicas, o que para fins do presente texto equivale a estabelecer um valor-verdade para cada fórmula, que pode ser verdadeiro ou falso[2].

O valor-verdade de uma fórmula é dado em relação a um modelo ou mundo possível. Por exemplo, "$x + y = 4$" tem valor-verdade 1 em um mundo no qual $x = y = 2$ mas tem valor-verdade 0 em um modelo no qual $x = y = 1$.

A maneira de formalizar a semântica e definir o significado na lógica foi proposta por Tarski em 1956, estabelecendo uma relação entre a linguagem e um modelo ou seja a descrição do mundo. Formalmente um modelo de Tarski

[2] Por questões notacionais utilizaremos 1 para verdadeiro e 0 para falso.

para uma linguagem lógica consiste de uma tupla $\langle \mathbf{O}, \mathbf{R}^n, \Phi \rangle$ onde $\mathbf{O}$ é um conjunto de objetos, $\mathbf{R}^n$ é um conjunto de propriedades e relações em $\mathbf{O}$ e Φ é uma função que atribui valor verdadeiro ou falso às fórmulas da linguagem como é o caso das tabelas-verdade dos operadores $\neg$, $\vee$, $\wedge$, e $\rightarrow$ na lógica proposicional.

3.3.3. Sistemas de Axiomas

Estes sistemas contêm certos axiomas[3] e um conjunto de regras de inferência a partir dos quais derivam-se consequências lógicas que são por sua vez utilizadas para gerar novas consequências até que uma determinada fórmula (que se deseja derivar) seja alcançada.

Um dos sistemas mais simples da lógica proposicional tem apenas 3 axiomas, a partir dos quais se pode gerar uma prova. Este sistema tem também 2 regras de inferência. A primeira é denominada *modus ponens* ou eliminação da implicação: $\phi, (\phi \rightarrow \psi) \vdash \psi$. A segunda é a regra de substituição, a qual permite substituir ϕ, ψ etc. por quaisquer wff's.

Na Seção 3.13 veremos a sintaxe, a semântica e os axiomas para lógicas modais utilizadas por agentes.

3.4. Resolução de problemas através de satisfação de restrições

Em IA, um foco tradicional é o da resolução de problemas (*problem-solving*). Um exemplo típico é encontrar uma sequência de ações que levam a estados desejáveis (um objetivo). Usualmente este tipo de problema se resolve através de algoritmos de busca, que no caso, podem ter um "*AI-flavor*", utilizando heurísticas como é o caso por exemplo do algoritmo A*.

Ainda que existam algoritmos heurísticos tratáveis, em geral resolução de problemas e otimização envolvem um espaço de estados muito grande e existe uma implicação na complexidade destes algoritmos (tempo e espaço). Desta forma, a comunidade de otimização combinatorial vem apelando para novas abordagens baseadas em agente que, supostamente, dividiriam o problema entre um conjunto de agentes cooperativos que resolveriam sub-partes em paralelo. Entretanto, existe aqui um trade-off não desprezível pois não está claro como o problema pode ser decomposto entre os vários agentes e como uma solução pode vir a ser construída de forma cooperativa. Por exemplo, existe um claro aumento da carga de comunicação. Entretanto, na prática esta carga tende a ser muito alta, possivelmente inviabilizando o seu uso em problemas do mundo real, *se o projeto dos agentes não for feito de forma cuidadosa*.

Uma abordagem para resolução de problema é via satisfação de restrições (em inglês[4] *constraint satisfaction problem* ou CSP). Problemas desta natureza

[3] Na verdade trata-se de *esquemas* de axiomas pois determinados símbolos como ϕ e ψ podem ser substituídas por fórmulas arbitrárias.

[4] Neste capítulo serão mantidos em inglês os termos abreviados que estão estabelecidos na literatura de tal forma.

podem ser resolvidos através de um algoritmo de busca que aproveita a estrutura dos estados para encontrar a solução de problemas grandes. Colocado de forma simples, um CSP é definido por um conjunto de variáveis e por um conjunto de restrições. Uma solução é uma atribuição de valores para cada variável que não viole as restrições.

CSP é um formalismo bastante utilizado na IA para representação simples e estruturada de problemas, como por exemplo escalonamento e alocação de tarefas, agendamento de reuniões, etc. Um CSP envolve três componentes: *variáveis*, *domínios* e *restrições*. Uma variável corresponde a uma parte do problema que pode ter seu valor alterado. Um domínio consiste de um conjunto de valores pré-definidos que uma variável pode assumir. Uma restrição corresponde a uma condição que deve ser satisfeita ao se atribuir valores para as variáveis. O objetivo em um CSP é definir um valor para cada variável de forma a satisfazer todas as restrições.

No entanto, determinados CSPs podem apresentar não apenas uma, mas várias soluções. Nestes casos é estabelecido um critério de qualidade para avaliar cada solução, e é preferível aquela com melhor qualidade. Esta qualidade normalmente é definida em termos de funções de custo, podendo-se optar por soluções que maximizem ou minimizem o custo dependendo do problema. Em outros casos, a situação oposta pode ocorrer, ou seja, não é possível satisfazer todas as restrições ao mesmo tempo. Quando isto ocorre, opta-se por uma atribuição de valores às variáveis que também maximizem ou minimizem uma função de custo especificada para o problema. Problemas que possuem estas características são denominados de COP (do inglês *Constraint Optimization Problems*).

3.4.1. Problema de Satisfação de Restrições

Em um CSP temos um conjunto $X = \{x_1, \ldots, x_n\}$ de variáveis. A cada variável x_i está associado um domínio D_i não vazio, finito e discreto. O valor que a variável assume é tomado de seu respectivo domínio. Além disto, há um conjunto de restrições $C = \{C_1, \ldots, C_m\}$. Cada restrição C_j envolve um subconjunto de variáveis de X e especifica as combinações de valores permitidas para o subconjunto em questão. Um estado do problema corresponde a uma *atribuição* de valores a algumas ou a todas as variáveis. Uma atribuição de um valor v_i para uma variável x_i é representada por $\langle x_i, v_i \rangle$. Um estado é portanto um conjunto de atribuições $\{\langle x_i, v_i \rangle, \langle x_j, v_j \rangle, \ldots\}$.

Uma restrição em um CSP é caracterizada pela quantidade de variáveis envolvidas. Uma *restrição binária* aplica restrições a duas variáveis. Em relação às variáveis, estas podem ser discretas com domínio finito, discretas com domínio infinito, ou contínuas (nos dois últimos casos, o domínio de cada variável possui infinitos valores). Neste capítulo será considerado apenas CSP (e extensões) com variáveis discretas e domínios finitos e restrições binárias.

Uma maneira de representar um CSP com restrições binárias é através de um grafo de restrições. Nesta forma de representação, os vértices do grafo cor-

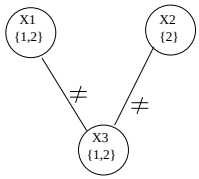

Figura 3.2. Exemplo de CSP

respondem às variáveis e as arestas correspondem às restrições. Um exemplo é apresentado na Figura 3.2, onde o objetivo é definir uma cor (representada numericamente) para cada variável ($\{x_1, x_2, x_3\}$) a partir das cores disponíveis nos domínios das variáveis ($\{1, 2\}, \{2\}, \{1, 2\}$), respeitando as restrições ($\{\{x_1 \neq x_3\}, \{x_2 \neq x_3\}\}$). A solução (neste caso, única) para o CSP é a atribuição $\mathcal{A} = \{\langle x_1 = 2\rangle, \langle x_2 = 2\rangle, \langle x_3 = 1\rangle\}$.

3.4.2. Problema de Otimização de Restrições

Quando um CSP apresenta mais de uma ou nenhuma solução que satisfaça todas as restrições, o objetivo passa a ser encontrar a *melhor* solução, o que aumenta o grau de complexidade do problema.

Além dos elementos que compõem um CSP (variáveis, domínios e restrições), em um COP temos um elemento adicional $\mathcal{F}(\mathcal{A})$, que corresponde a uma função objetivo que mapeia cada solução (conjunto de atribuições) para um valor numérico. Esta função objetivo é normalmente uma agregação de funções de custo f, que são análogas às restrições de um CSP, com a diferença de que, ao invés de serem booleanas (*satisfeita* ou *não satisfeita*), retornam valores numéricos. A solução de um COP corresponde então a uma atribuição completa $\mathcal{A}^*$ cujo valor da função objetivo é ótimo.

Um exemplo de COP é apresentado na Figura 3.3. Ele consiste em associar um valor para cada uma das variáveis, respeitando as restrições e de forma a *minimizar* a função objetivo $\mathcal{F}(\mathcal{A})$ dada pela agregação (soma) das funções de custo f, ou seja, $\mathcal{F}(\mathcal{A}) = f(x_1, x_3) + f(x_2, x_3)$. A solução para este COP é a atribuição completa $\mathcal{A}^* = \{\langle x_1 = 2\rangle, \langle x_2 = 2\rangle, \langle x_3 = 2\rangle\}$.

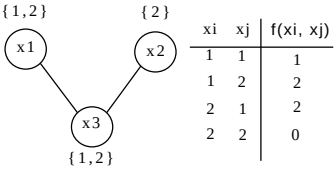

Figura 3.3. Exemplo de COP

Como será visto na parte III (Seção 3.14), um dos grandes problemas destas abordagens, seja CSP ou COP, é a hipótese do conhecimento total do problema, que é uma hipótese forte atualmente já que, na Internet, segurança e privacidade são cruciais. Existem situações em que o conhecimento a respeito do problema, não é ou não pode ser centralizado. Para especificar um CSP distribuído, serão vistos os formalismo DisCSP e DCOP. Ambos tratam de problemas em ambientes distribuídos e utilizam agentes para representar e controlar as variáveis, configurando-se um sistema multiagente.

3.5. Planejamento e tomada de decisão

Nesta seção será introduzido o problema de planejamento, que pode ser definido como a tarefa de apresentar uma sequência de ações que podem alcançar um dado objetivo. Este problema é central em IA pois diversas atividades do dia-a-dia envolvem planejamento para tomada de decisão. Como um simples exemplo temos a decisão que envolve ir de Porto Alegre a Tóquio para participar de uma conferência. Um agente reativo tomaria o primeiro avião saindo da cidade origem e iria adaptando-se pelo caminho até, eventualmente, chegar ao destino. Um agente que planeja consultaria um agente de viagem ou a Internet e veria rotas, horários, etc. e faria um planejamento.

Planejamento em IA pode ser visto como fazer programação automática (no estilo de prolog): um algoritmo de planejamento recebe como entrada uma descrição de um conjunto de ações possíveis bem como os efeitos que estas ações causam. O algoritmo então desenvolve um plano (um programa) que descreve como as ações possíveis podem ser executadas de modo a se chegar ao estado-objetivo.

Esse processo envolve os seguintes passos: i) representar o estado atual do mundo (ambiente); ii) representar o estado desejado do mundo; iii) representar um conjunto de ações atômicas que podem ser executadas; iv) determinar, através de um resolvedor de problema, um plano que, composto de ações atômicas, leve o estado do mundo da configuração inicial à configuração final desejada (ou vice-versa).

A abordagem mais conhecida que implementa estes passos é o STRIPS [Fikes and Nilsson 1971]. Deve-se a este trabalho a ideia de descrever ações e efeitos. Além disto, STRIPS é baseado em lógica de primeira ordem com formalismo de representação simbólica e como mecanismo de apoio para tomada de decisão de forma dedutiva através de mecanismos de prova e cálculo situacional.

Para efeito do presente texto, consideraremos três tipos de abordagens, as quais são definidas com base no tipo de ambiente no qual a tarefa de planejamento se desenvolve. No planejamento clássico, o ambiente é estático, determinístico e completamente observável (no sentido de que o planejador tem acesso a todas as variáveis relevantes). Já no planejamento não clássico, o ambiente geralmente é parcialmente observável e portanto estocástico. Por fim, planejamento multiagente trata do problema no qual vários agentes pre-

cisam cooperar ou pelo menos se coordenar. Este último será discutido na Seção 3.15.

3.5.1. Planejamento clássico

Em STRIPS, o estado inicial é representado através de conjunções de predicados. Por exemplo, $In(airplane_1, POA) \wedge In(airplane_2, BSB)$ representa um estado no qual o avião 1 está em Porto Alegre e o avião 2 em Brasília. Os objetivos são representados de forma semelhante como por exemplo: $In(airplane_1, BH)$ As ações possíveis são especificadas em termos de pré-condições, as quais devem ser válidas para que possam ser executadas, bem como em termos de efeitos que resultam da sua execução, como por exemplo:

$Action(Fly(p, from, to))$
$PRECOND : In(p, from) \wedge Airplane(p) \wedge Airport(from) \wedge Airport(to)$
$EFFECT : \neg In(p, from) \wedge In(p, to)$

No presente texto nos deteremos apenas nesta ideia básica do STRIPS e não entraremos em detalhes relacionados aos problemas do enquadramento e outros por não serem relevantes para a discussão contida na Seção 3.15.

3.5.2. Planejamento não clássico: tomada de decisão em ambientes estocásticos

O planejamento clássico não trata ambientes dinâmicos embora estes sejam na verdade aqueles que encontramos na maioria dos problemas práticos. Desta forma, a maioria das aplicações que se pretendam realistas não são adequadamente tratadas pelo planejamento clássico. Para representar e tratar ambientes estocásticos, o modelo básico é o que modela transições (de estados) em sistemas dinâmicos. A representação é através de cadeias de Markov enquanto que o processo de planejamento em si (ou seja qual ação realizar em cada estado) é tratado pelo formalismo denominado processo de decisão de Markov ou MDP (do inglês *Markov decision process*)[5], que é a formalização de um processo decisório sequencial que se desenrola em um ambiente completamente observável e considera um único agente. Um MDP é a principal ferramenta para que um planejador modele e decida sobre ações em um sistema dinâmico.

Formalmente um MDP$= (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ consiste de um conjunto de estados $\mathcal{S}$, de ações $\mathcal{A}$, da recompensa imediata esperada $\mathcal{R}^a_{s,s'}$ obtida quando o agente transiciona do estado s para o estado s' realizando a ação a, bem como do conjunto das probabilidades de transição $\mathcal{P}$.

Em suma, a cada instante de tempo t, o MDP está num estado s_t. O agente realiza a ação $a_t \in \mathcal{A}$ que causa uma transição para o estado s_{t+1} com probabilidade $\mathcal{P}^a_{s_t,s_{t+1}}$. O agente então recebe uma recompensa $r_{t+1} \in \Re$. A propriedade de Markov implica que a probabilidade de alcançar o estado

[5] Esta formalização não é a única mas para o caso monoagente é a mais relevante.

s_{t+1} (e receber r_{t+1}) depende apenas do estado s_t i.e. esta probabilidade é independente de $s_{t'}$, $a_{t'}$, $r_{t'+1}$ para todo $t' < t$.

Devido à propriedade de Markov é possível definir uma política π que é função apenas do estado atual do sistema e não do histórico. π é um mapeamento de estados em ações e portanto uma forma de plano (condicional), no qual uma ação ramifica o estado do sistema. MDP's são resolvidos de forma geral utilizando programação dinâmica.

Como isto se aplica a um agente planejador? O agente é modelado em termos da política π. Uma política determinística $\pi = \mathcal{S} \rightarrow \mathcal{A}$ é um mapeamento de um conjunto de estados para um conjunto de ações. De posse desta política, o agente deve aplicá-la selecionando a ação $\pi(s)$ quando no estado s. Uma política determinística é um caso especial de uma política estocástica. Esta última especifica uma *distribuição* de probabilidade sobre $\mathcal{A}$ para cada estado s ($(\pi(s, a)$ denota a *probabilidade* de escolher uma ação a no estado s).

Os dois principais problemas aqui associados são o fato de que um MDP assume que o ambiente é totalmente observável pelo agente, além de ser um formalismo monoagente. Na Seção 3.16 serão discutidas as extensões possíveis para se contornar o segundo destes problemas.

Quanto à hipótese de que em um MDP o agente recebe ou percebe informação *completa* sobre o estado do ambiente, esta hipótese é muito forte e não casa com o que se encontra na maioria dos problemas do mundo real.

Desta forma, uma extensão de MDP denominada POMDP (partial observable MDP) é definida pela tupla $(\mathcal{S}, \Omega, o, \mathcal{A}, \mathcal{T}, \mathcal{R})$ que contem dois subconjuntos além dos já definidos. Ω é o conjunto de observações e o é uma função que mapeia cada estado a uma observação. Desta forma o agente não recebe (ou percebe) necessariamente o estado correto mas sim realiza uma observação a qual é mapeada para um estado (que pode não ser o correto).

Infelizmente foi provado [Littman 1994b] que encontrar uma política ótima para um POMDP é um problema NP-difícil. Existem alguns algoritmos baseados em valores de função (como para um MDP) mas eles são aplicáveis apenas a problemas muito pequenos (poucos estados, poucas ações, etc.) o que prejudica sua utilização prática.

A despeito deste fato, MDP continua a ser um formalismo popular pois é uma forma natural e concisa para representar sistemas estocásticos, evitando as dificuldades associadas aos formalismos baseados em cálculo situacional como STRIPS. Além disto, MDP é bastante popular em aprendizado conforme discutido na próxima seção, bem como em aprendizado multiagente, onde os MDP's necessitam ser estendidos conforme apresentado na Seção 3.16.

3.6. Aprendizado

Aprendizado tem um papel fundamental em qualquer sistema que se pretenda inteligente. Ele pode ser tratado de forma genérica como a área que desenvolve algoritmos que aumentam a habilidade do agente de casar uma série de padrões de entrada com suas correspondentes saídas.

Embora existam vários métodos de aprendizado (supervisionados e não supervisionados), aquele que tem se destacado na área de agentes é o aprendizado por reforço (doravante abreviado por RL do inglês *reinforcement learning*). A razão disto é clara: embora métodos supervisionados e não supervisionados sejam importantes, eles não são adequados para aprendizado a partir de interação (por exemplo, com um ambiente). Comparando-se por exemplo RL a aprendizado supervisionado (onde instâncias de treinamento indicam o comportamento correto em situações particulares), pode-se dizer que RL é mais geral e mais difícil, dado que a tarefa do aprendizado baseia-se em menos conhecimento. Portanto, como RL parece ser a forma mais adequada para o caso de um agente interagindo com um ambiente, esta seção se concentra nesta técnica, aqui para o caso monoagente.

Aprendizado por reforço se ocupa de um tipo específico de aprendizado de máquina ou seja o aprendizado do mapeamento *situação-ação* de modo a maximizar um sinal numérico chamado de recompensa[6].

No RL não se diz explicitamente ao aprendiz ou agente[7] o que fazer ou que ação realizar. O próprio agente deve, por tentativa-e-erro, encontrar a ação que traz a maior recompensa. RL é portanto uma área de interface entre ciência da computação, pesquisa operacional e engenharia de controle.

RL é uma técnica usada para resolver a seguinte classe de problemas: dado um agente (robô, animal) situado em um ambiente, sua percepção deste ambiente, as ações por ele realizadas e a recompensa percebida, como planejar a estratégia ótima ou em outras palavras, como encontrar a política que maximiza a recompensa a longo prazo?

Notar que nesta classe de problemas, a percepção que o agente tem do ambiente pode ser limitada. Além disto, a recompensa que o agente recebe pode ser interpretada como um sinal emitido pelo ambiente o qual reflete quão boa ou ruim foi a ação. Ocorre que este sinal pode ser esparso, atrasado ou apresentar ruídos.

Um exemplo de recompensa atrasada (e também esparsa) é uma partida de futebol pois o ganho de um jogador apenas é atribuído ao final da partida. Desta forma o jogador pode apenas relacionar a recompensa com uma determinada ação (ou, mais comumente neste caso, uma sequência de ações) como por exemplo um passe, ao final da partida quando o escore final é estabelecido. Este é obviamente um exemplo exagerado pois há formas de contornar este problema. Porém, ele serve como ilustração.

[6] Por questão de uniformidade, neste texto será usado o termo recompensa para denotar o que na literatura aparece, segundo o contexto e a fonte, como reward ou payoff, e que na literatura em português pode vir traduzido como recompensa ou ganho; notar ainda que, apesar do nome, recompensa pode indicar uma quantidade tanto positiva como negativa.

[7] Neste texto será empregado o termo agente com sentido de aprendiz ou jogador; estes últimos são mais comuns nas áreas de aprendizado de máquina e teoria de jogos, respectivamente.

3.6.1. Aprendendo a política ótima: caso monoagente

Como já foi mencionado na Seção 3.5.2, um agente é modelado em termos de uma política π. A tarefa do agente que utiliza RL consiste em usar as recompensas recebidas para aprender uma política ótima para aquele ambiente em particular.

Até aqui assumimos que o agente tem um modelo completo do ambiente (a variável $\mathcal{P}$) e conhece a função de recompensa $\mathcal{R}$ da definição de MDP vista. Esta hipótese não é razoável em domínios complexos. Desta forma, vamos dotar o agente de ferramentas com as quais ele irá *aprender* a se comportar com sucesso no ambiente.

Para tanto, é interessante que o agente consiga calcular uma política ótima ou seja, em $t = 0$, encontrar a política π^* que maximiza a soma acumulada de ganhos ao longo do tempo. Esta soma é denotada $R_t = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$ onde $\gamma \in [0, 1]$ é o chamado fator de desconto que determina quanto se deve ponderar os ganhos imediatos (a cada t) e, consequentemente, quanto se deve ponderar os ganhos futuros. $\gamma < 1$ garante que R_t é finito. Dado que as transições de estado são aleatórias, o ganho acumulado (e descontado) R_t pode ser diferente do ganho *esperado* $\mathbf{E}[R_t|\pi, s]$ que um agente obtem *em média* se aplicar a política π no estado inicial s_0. O valor esperado $V^\pi(s)$ do estado s é dado pela Eq. 1, a equação de Bellman para V^π. O agente deve entretanto aprender a encontrar a política ótima ou seja aquela que lhe traz alta recompensa a longo prazo.

$$V^\pi(s) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t \, r_{t+1} | \pi, s_0 = s\right] = \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{s,s'} \left(\mathcal{R}^a_{s,s'} + \gamma \, V^\pi(s')\right) \tag{1}$$

É possível obter-se uma ordenação parcial do conjunto de políticas (e.g. $\pi_1 > \pi_2$ se $V^{\pi_1}(s) > V^{\pi_2}(s)$ para todo $s \in \mathcal{S}$) o que nos permite encontrar π^*, a melhor política ou política ótima (que sempre existe mas pode não ser única). π^* está associada à função ótima de valor de estado V^*, que é definida como na Eq. 2.

$$V^*(s) = \max_\pi V^\pi(s), \ \forall s \in \mathcal{S} \tag{2}$$

Vários métodos podem ser empregados para se aprender a política ótima. [Sutton and Barto 1998] descrevem três classes de métodos para resolver problemas de RL: programação dinâmica, métodos de Monte Carlo e aprendizado via diferença temporal. O primeiro tem a vantagem do formalismo matemático ser bem desenvolvido mas requer que o agente tenha um modelo do ambiente. Métodos de Monte Carlo não requerem tal modelo, são conceitualmente simples mas não são adequados para o cálculo associado com aprendizado incremental. Por fim, métodos de diferença temporal são incrementais, não

requerem o modelo do ambiente mas são mais complexos de analisar. Exporemos a seguir alguns dos métodos associados a aquelas classes; a discussão sobre as versões multiagente dos métodos mais usados aparecem na Seção 3.16.3.

3.6.1.1. Iteração de valor

Da equação de Bellman (Eq. 1), dado um espaço finito de estados, é possível se obter um conjunto de cardinalidade $|\mathcal{S}|$ de equações lineares, as quais podem ser resolvidas de forma geral por programação dinâmica. Dado um valor arbitrário para V, aplica-se este valor iterativamente ao lado direito da Eq. 1 até se encontrar V^*, o ponto fixo. Uma vez encontrado V^*, é possível encontrar $\pi^*(s)$ quando se escolhe a ação que leva ao estado sucessor s' cujo valor é máximo ou seja:

$$\pi^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left(\sum_{s' \in \mathcal{S}} \left(R^a_{s,s'} + \gamma \times \mathcal{P}^a_{s,s'} \times V^*(s') \right) \right)$$

Este método tem uma desvantagem: exige que $\mathcal{P}$ seja conhecida o que é uma hipótese muito forte. Desta forma, não nos ocuparemos mais deste método neste texto.

3.6.1.2. Diferença temporal e Q-learning

Para endereçar o problema de que $\mathcal{P}$ pode não ser conhecida, foram propostos métodos que estimam os valores V a partir de grandezas que são necessariamente observáveis como a recompensa r nos estados s e s', supondo que estes estados são observáveis com 100% de confiabilidade [8].

Estes métodos que dispensam o conhecimento de $\mathcal{P}$ são chamados de métodos livres de modelo (*model-free*). Eles aproximam o valor da ação, denotado $Q(a)$ ou de forma mais geral $Q(s,a)$ ou seja o valor da ação a quando o agente a realiza no estado s.

O algoritmo Q-learning constrói, iterativamente, uma sequência de valores Q. No instante t o agente observa seu estado (s), realiza uma ação (a), transiciona para o estado s' e recebe r como recompensa. A partir desta observação ele atualiza $Q(s,a)$ da seguinte forma:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \, max_{a'} \, Q(s',a') - Q(s,a) \right) \quad (3)$$

Na equação 3, $\alpha \geq 0$ é a taxa de aprendizado (que pode decrescer com o tempo) e γ é um fator de desconto. Foi provado [Watkins 1989] que, no limite de

[8] É possível uma generalização para o caso de observação parcial apresentado na Seção 3.5.2.

um conjunto infinito de observações de transições de estados, a convergência (para o valor ótimo Q^*) é garantida, desde que todos os pares estado-ação (s, a) sejam visitados infinitamente e frequentemente e que algumas restrições em relação à α sejam obedecidas.

A vantagem deste método é que dispensa o conhecimento de $\mathcal{P}$. Uma vez que $Q^*(s, a)$ é conhecido, o agente pode usar uma política gulosa que seleciona a ação a que maximiza $Q^*(s, a)$ no estado s, para todo $s \in \mathcal{S}$.

Uma desvantagem própria de Q-learning é que a taxa de convergência pode ser baixa já que apenas um par estado-ação é atualizado a cada passo. Para mitigar este problema pode-se usar métodos baseados em traços de elegibilidade como $Q(\lambda)$, SARSA(λ) e TD(λ). Neste texto não nos deteremos nestes métodos. O leitor pode consultar [Sutton and Barto 1998] para mais detalhes. Apenas mencionaremos que traços de elegibilidade se baseiam na seguinte ideia: dada uma ação a em s, o valor Q da função é atualizado não apenas para o estado s mas sim para todos os estados que ocorreram na trajetória que levou a s.

3.6.1.3. Aproximação de funções

O requisito de visitação de pares estado-ação nos leva à ideia de que estes valores podem ser armazenados em uma tabela (onde s e a são as entradas e Q é o valor correspondente). Entretanto, se a cardinalidade do conjunto de combinações estado-ação for muito grande ou se os valores forem contínuos (infinitos), tal representação não é viável.

Para tratar este problema, os valores da funções (por exemplo Q) podem ser representados em termos de uma função do tipo linear ou do tipo perceptron multi-camada, funções estas que são parametrizadas. Detalhes deste processo podem ser encontrados em [Sutton and Barto 1998].

3.6.1.4. Métodos baseados em modelo

Os métodos de RL baseados em modelos assumem que a função de transição $\mathcal{P}$ e a função de recompensa $\mathcal{R}$ são disponíveis. Um algoritmo desta classe é o *Prioritized Sweeping* (PS) que é semelhante ao Q-Learning, exceto por manter uma estimativa do modelo de comportamento do ambiente. Além disso, ele atualiza mais de um valor Q por iteração. Os valores Q que devem sofrer atualização a cada iteração são determinados por uma fila de prioridades de atualização. A cada iteração, são feitas novas estimativas acerca do funcionamento do ambiente.

Um dos problemas com este método é que ele assume um número fixo de modelos de comportamento do ambiente. Uma vez que essa suposição nem sempre é realista, o método denominado RLCD (*reinforcement learning with context detection*) [Silva et al. 2006] baseia-se na construção incremental de modelos. A cada modelo, é associada uma política ótima, que consiste em um

mapeamento (a ser aprendido) entre estados e ações. Um contexto consiste de um MDP com funções de transição e de recompensa distintas. O mecanismo para detecção de contextos utiliza um conjunto de modelos parciais para efetuar a previsão da dinâmica do ambiente. Cada modelo parcial m contem uma função $\hat{\mathcal{P}}_m$, que estima probabilidades de transição, e também uma função $\hat{\mathcal{R}}_m$, que estima as recompensas que podem ser esperadas em cada situação. A fim de detectar as mudanças de contexto, o método mensura o quão bem o modelo de ambiente atualmente em uso consegue prever o que realmente está sendo observado. Quando não existirem modelos com qualidade superior à qualidade mínima (o que significa que nenhum modelo existente é capaz de explicar o estado observado), um novo modelo é construído incrementalmente.

3.6.2. Problemas associados ao aprendizado por reforço

São 3 os problemas comumente encontrados por quem modela um problema de RL. O primeiro está ligado à atribuição temporal da recompensa (*temporal credit assignment*). Como já foi dito, a recompensa pode ser atribuída (ou percebida) com atraso de modo que ela pode não ser necessariamente relacionada com a última ação realizada. O segundo problema é relacionado com o espaço de pares estado-ação. Os algoritmos de RL em geral exigem a visitação frequente de cada um destes pares. Este problema é extremamente agravado em cenários multiagentes como será visto na Seção 3.16. O terceiro problema está ligado ao dilema exploração-aproveitamento (ou exploração-desfrute) que ocorre porque uma política é aprendida pelo agente ao mesmo tempo em que este explora seu ambiente.

Existe uma política ótima de exploração? Desta questão trata o problema denominado *n-armed bandit*. Este problema aparece na literatura como uma metáfora para problemas de escolha de ação via experimentação e subsequente recompensa. Um exemplo seria um médico decidindo qual tratamento administrar a pacientes; a recompensa seria a cura ou piora no estado do paciente.

O nome *n-armed bandit problem* vem da analogia com máquinas caça-níqueis na qual se tem n alavancas e o agente deve escolher uma alavanca a cada turno, com o objetivo de maximizar o ganho. Cada alavanca tem uma recompensa média associada a ela, mas este valor não é conhecido pelo agente. Se fosse, bastaria o agente selecionar a de maior valor esperado. Desta forma, o agente deve estimar estes valores através de experimentação ou seja de exploração.

Se o agente puxar uma determinada alavanca e obtiver $r = 50$ por exemplo, ele pode considerar este valor suficiente e continuar desfrutando dele selecionando esta ação para sempre. Neste caso, como r é probabilístico, o agente pode vir a ter ganhos iguais, maiores, ou menores. Em cada um destes casos ele pode optar por continuar puxando a mesma alavanca ou não. De qualquer forma, este procedimento não explora suficientemente seu espaço de ações. O agente poderia, por exemplo, optar por puxar todas as alavancas pelo menos

uma vez.

Idealmente o agente deveria puxar cada alavanca um número grande de vezes a fim de obter uma estimativa da distribuição de probabilidade associada a cada alavanca. Entretanto, cada vez que o agente realiza tal ação há um custo. É relativamente claro que deve existir um compromisso entre exploração e aproveitamento. Dependendo deste custo, o agente pode explorar mais ou menos seu ambiente para, posteriormente, desfrutar do que aprendeu passando a escolher a alavanca que lhe traga o maior ganho.

Parte II: Agentes Autônomos e Sistemas Multiagentes

Nesta parte serão introduzidos os conceitos relativos a: agentes e sistemas multiagente; agentes como sistemas dedutivos; agentes como sistemas reativos; coordenação, cooperação, negociação e comunicação em sistemas multiagentes.

Ao final desta parte, os leitores serão confrontados com duas instâncias paradigmáticas da IA – sistema especialista e futebol de robôs – a fim de refletir sobre as diferenças causadas pelo fato de que no primeiro caso trata-se não apenas de um único agente, mas também de um agente não situado em qualquer ambiente. Já no segundo caso, há questões complexas envolvidas como: agentes são situados em um ambiente, agentes devem colaborar entre si enquanto time, e agentes competem entre si enquanto oponentes.

3.7. IA Distribuída: histórico e taxonomia

Há algumas décadas a IA tem focado o emprego de suas técnicas de resolução de problemas em uma entidade única, seja ela um robô, um sistema especialista, um veículo, etc. Entretanto, a rigor, nenhuma destas entidades deveria ser tratada de forma isolada pois, em geral, trata-se de uma associação de unidades que podem ter complexos inter-relacionamentos. Por exemplo, para ser capaz de abrir uma porta, um robô tem que realizar pelo menos duas tarefas: reconhecer e processar uma imagem e realizar um planejamento das ações, sendo ambas afetadas pela dinâmica do ambiente e pela presença de outros agentes. Em aplicações reais, tais entidades não podem ser consideradas como uma unidade sob pena de se obter um sistema cuja complexidade chegue a um nível intratável.

Por outro lado, ao se decompor um determinado problema, são geradas várias partes que interagem umas com as outras. Tal interação deve ser cuidadosamente tratada sob pena de se ter que lidar com complexidade igualmente alta. Esta foi a motivação inicial que levou ao aparecimento de uma nova área da IA, inicialmente denominada “inteligência artificial distribuída” (IAD).

Segundo [Bittencourt 2001], “a IAD é uma sub-área da IA que estuda o conhecimento e as técnicas de raciocínio que podem ser necessárias ou úteis para que os agentes computacionais participem de sociedades de agentes“. A IAD se preocupa com uma ou mais dentre as seguintes tarefas: decomposição de problemas complexos, alocação de tarefas dentre um grupo de agentes de forma que estes melhorem seu desempenho como grupo, distribuição do controle (das tarefas), evitar interações danosas, comunicação, síntese das soluções parciais ou locais e garantir a solução global do problema, se possível de forma cooperativa. É preciso frisar aqui que tal forma cooperativa não necessariamente significa que os agentes tenham que se comportar de maneira benevolente ou altruísta; eles podem ser levados a ter comportamentos antagônicos. Neste caso, a cooperação pode emergir quando existe a necessidade dos agentes trabalharem juntos a fim de levar a cabo seus objetivos privados ou particulares. Na literatura, tais agentes também são denominados intencionais ou ainda agentes motivados individualmente ou auto-motivados (ou seja motivados por seus objetivos ou por seus estados internos). IAD também é normalmente associada com o termo sociedade de agentes, o que aqui significa uma rede de agentes na qual cada um tem habilidades particulares mas não é capaz de resolver o problema como um todo devido à falta de recursos, informação ou perícia.

Originalmente, as várias técnicas de IAD relatadas na literatura foram divididas em duas grandes áreas de acordo com sua visão: orientadas ao problema (granulometria alta) e orientadas ao agente (granulometria fina). Posteriormente, estas duas visões tornaram-se conhecidas como “resolução distribuída de problemas” (DPS, de Distributed Problem Solving) e sistemas multiagentes.

DPS ocupa-se preponderantemente com a decomposição de um problema entre uma rede de agentes e com os mecanismos de cooperação e comunicação necessários para resolver o problema.

Já um sistema multiagente ocupa-se principalmente com a coordenação dos agentes, especialmente se forem individualmente motivados. Neste caso, normalmente os agentes devem compartilhar intenções, planos e conhecimento de forma a poder se coordenar. Coordenação é necessária para que se resolvam conflitos que podem surgir quando se alocam recursos limitados ou quando os agentes têm preferências conflitantes. Assim, coordenação depende muito da forma de interação que emerge dos agentes. Para ser efetiva, é preciso que haja um certo grau de previsão das atitudes dos demais agentes, o que pode ser obtido através de um modelo das preferências dos outros agentes, por exemplo.

Devido ao relativo baixo interesse atual em torno de DPS (compreensível se verificarmos o seu reduzido escopo frente aos problemas atualmente colocados por exemplo pela Internet), o restante desta discussão será focado em sistemas multiagentes. Antes porém, serão introduzidos os conceitos de agente e algumas de suas características.

3.8. Agentes autônomos

A palavra agente vem do latim *agere* ou agir, porém o termo agente não tem uma definição que seja amplamente aceita. Para [Wooldridge 2002] "um agente é um sistema computacional situado em algum ambiente, sendo capaz de realizar, de forma independente (autonomante), ações neste ambiente, descobrindo o que necessita ser feito para satisfazer seus objetivos, ao invés de ter que receber esta informação". Para Franklin e Graesser, "um agente autônomo é um sistema situado dentro de e parte de um ambiente, ambiente este que o agente percebe e nele age ao longo do tempo, perseguindo um objetivo dentro de sua própria agenda. A ação do agente por sua vez afeta o que ele perceberá no futuro".

Nestas duas definições aparecem alguns conceitos importantes; agentes devem:

- estar situados (em um ambiente)
- descobrir autonomamente o que fazer (sem ser dito)
- perseguir sua própria agenda
- agir ao longo do tempo

Como discutiremos na Seção 3.11, estas características constituem um marco delimitador entre técnicas e sub-áreas da IA.

Agentes podem ser ter uma arquitetura reativa ou deliberativa (também chamada cognitiva). Um agente reativo é baseado em modelos simples como o estímulo-resposta e baseado em arquiteturas de subsunção conforme proposto por [Brooks 1986]. De fato, uma das implicações práticas desta arquitetura foi que a área de agentes se tornou um campo fértil para teste em cenários realistas. Agentes reativos são usados principalmente em ambientes onde eles são numerosos (ordem de centenas, milhares de agentes) e a eficiência do processamento é uma questão chave. Um exemplo típico de agente reativo é aquele com inspiração em paradigmas da etologia (que estuda o funcionamento de insetos sociais e sua organização).

Agentes deliberativos são inspirados em organizações sociais entre seres humanos como empresas e suas hierarquias organizacionais, comunidade científica e mercados no sentido da economia. Neste caso a arquitetura do agente representa explicitamente não apenas o ambiente mas também os demais agentes. Neste tipo de arquitetura o planejamento das ações futuras é em geral no estilo do planejamento clássico visto na Seção 3.5 (considerando extensões para o caso multiagente conforme a Seção 3.15). Isto envolve portanto uma caracterização formal de atitudes mentais como intenções, objetivos, crenças, as quais serão vistas na Seção 3.13. Além disto a comunicação entre agentes tem um papel importante neste tipo de arquitetura, o que não é o caso em uma arquitetura reativa. Justamente devido à complexidade das arquiteturas deliberativas, este tipo de agente é encontrado em cenários onde seu número é da ordem de dezenas de agentes.

Outra característica, menos óbvia e mais polêmica, é que um agente pode ser analisado sob uma perspectiva microeconômica, ou seja, pode ser visto como uma entidade racional, auto-motivada e não disposta a mostrar benevolência perante os outros. Os seguidores desta linha colocam a diferença entre agentes e objetos da seguinte maneira: enquanto agentes fazem algo para obter um certo ganho, objetos o fazem sem tal contrapartida. Esta visão de agência é criticada por ser orientada ao ganho e por não dar muito espaço para que o agente coopere com outros ao menos que isso contribua para que seus objetivos sejam satisfeitos. Entretanto, tal abordagem é válida e útil em certos domínios onde os objetivos dos agentes são altamente antagônicos.

3.9. Sistemas multiagentes

Assim como não há uma definição universalmente aceita sobre o que é um agente, também não existe uma definição própria para sistemas multiagentes. Pode-se dizer que um sistema multiagente é um sistema que consiste de um número de agentes que interagem uns com os outros. Além disto, para interagir de forma eficaz, os agentes deste sistema devem ser capazes de cooperar, se coordenar e negociar entre si [Wooldridge 2002].

Um sistema multiagente possui algumas características, como por exemplo: cada agente tem informação ou capacidades incompletas, ou seja, visão limitada para resolver o problema; não existe um controle global do sistema; o conhecimento se encontra distribuído.

Uma das motivações para o desenvolvimento de sistemas multiagentes é a possibilidade de resolver problemas que não podem ser resolvidos de forma centralizada (por limitação de recursos ou desempenho). Por outro lado, como os sistemas multiagentes prevêem vários agentes envolvidos em um ambiente normalmente complexo, conflitos precisam ser tratados ou em fase de projeto ou durante a execução das tarefas.

3.10. Coordenação e cooperação em sistema multiagente

3.10.1. Coordenação de agentes

De uma forma geral, a coordenação aumenta o desempenho dos sistemas multiagentes e é um componente chave, uma vez que cada agente possui apenas uma visão local e incompleta. Ela é fundamental em tarefas de planejamento conforme detalhado no capítulo 9 de [Wooldridge 2002], no capítulo 3 de [Weiss 1999] e resumido na Seção 3.15 do presente texto. Por fim, coordenação é necessária a fim de se resolver conflitos, alocar recursos limitados, conciliar preferências e buscar soluções de caráter global.

Devido à importância da coordenação, é conveniente se olhar para seu significado em outras áreas. De fato, é possível encontrar vários trabalhos envolvendo o estudo de coordenação sob o aspecto multidisciplinar (computação, teoria de organizações, administração, economia, linguística e psicologia). Um estudo interessante é o de [Malone and Crowston 1994] que discute a seguinte questão: como o uso de tecnologia de informação muda a forma como as pes-

soas colaboram? Neste trabalho são relacionadas várias definições para coordenação, entre as quais as mais relevantes para a área de sistemas multiagentes são: i) coordenação é a tarefa de gerenciar dependências entre atividades [Malone and Crowston 1994]; ii) coordenação é a decisão sobre como compartilhar recursos, a qual em geral envolve negociação e comprometimentos [Rosenschein and Zlotkin 1994].

Desta forma fica claro que o projeto de mecanismos adequados de coordenação é uma tarefa fundamental na concepção de um sistema multiagente. Neste texto não serão fornecidos detalhes sobre os mecanismos em si por serem de diversas naturezas. O leitor pode consultar [Weiss 1999] ou [Wooldridge 2002] para este fim. Nos limitaremos a elencar os principais mecanismos propostos: coordenação através de revisão das ações, teoria de jogos, sincronização, negociação, rede de contratos, quadro-negro, troca de modelos sobre preferências.

3.10.2. Cooperação entre agentes

Em uma sociedade ou rede de agentes onde cada um tem conhecimento e perícia limitada, a cooperação é necessária para que o objetivo maior de toda a sociedade seja atingido. O nível e forma de cooperação entre agentes pode variar em um espectro que vai desde totalmente cooperativa até antagônica. Sistemas totalmente cooperativos em geral pagam um alto preço sob a forma de custos de comunicação. Enquanto agentes totalmente cooperativos podem trocar objetivos entre si de forma a solucionar o problema como um todo, em sistemas onde os agentes são antagônicos, estes podem optar por não cooperar de modo algum, além de inclusive tentar impedir as ações dos demais agentes se estas estão em conflito com o objetivo do agente em questão. Se por um lado aqui os custos de comunicação são baixos (ou inexistentes), por outro lado a eficiência pode ser seriamente comprometida.

O papel da cooperação difere nas diversas abordagens propostas. Enquanto a cooperação pode ser vista como um caso especial de coordenação entre agentes não antagônicos (ou seja cooperativos por natureza), alguns pesquisadores defendem que a cooperação pode emergir também entre agentes antagônicos, desde que o benefício de tal cooperação aumente a chance do indivíduo atingir seu objetivo. Por exemplo, duas companhias telefônicas podem fazer um acordo a fim de rotear pacotes que sejam do interesse de ambas [Rosenschein and Zlotkin 1994]. Esta abordagem (e as inúmeras que dela derivaram) é baseada em ideias advindas da teoria de jogos (cooperativa e não cooperativa), enquanto formalismo matemático para analisar a natureza das interações e cooperações entre agentes antagônicos em sistemas multiagentes.

Para se conseguir raciocinar sobre cooperação, é preciso modelar as intenções dos demais agentes, o que pode ser feito de várias formas. A modelagem explícita dos estados mentais dos agentes é a forma mais sofisticada e será discutida na Seção 3.13.

Uma outra forma é via simples troca de informação entre os agentes. Normalmente este processo se apoia nas seguinte etapas: desenvolvimento de um plano que considere o comportamento dos outros agentes, comunicação das partes relevantes do plano, e comunicação do comprometimento de cada agente com suas ações. Dependendo do número de agentes envolvidos, da complexidade dos planos e a da precisão desejada, os custos de comunicação podem ser inviáveis.

Este fato motivou uma terceira linha de abordagens, a baseada em teoria de jogos, conforme já mencionado. Esta linha se apoia no conceito de conhecimento comum (*common knowledge*, que será explorado na Seção 3.13.2), aqui no sentido mais restrito empregado pelos formuladores daquela teoria. A idéia básica é que as crenças e intenções dos agentes são de conhecimento comum e enquanto este conhecimento for válido, os agentes não necessitam comunicação para cooperarem.

Resumindo, cooperação está associada com o compartilhamento de objetivos, enquanto que coordenação está associada com o fato de se considerar os planos dos outros. Desta forma, a menos que os agentes garantidamente tenham o mesmo objetivo, o uso de cooperação apenas não é efetivo; as ações dos indivíduos devem também ser coordenadas.

3.11. IA clássica versus IA multiagente: sistemas especialistas versus robótica

Dois objetos clássicos da IA são os sistemas especialistas e a robótica. Nesta seção discutiremos por que estas duas áreas situam-se de lados opostos no que se refere a paradigmas baseados em agentes. Lembramos que, nas definições de agente vistas, algumas características são sumamente importantes: estar situado (em um ambiente); decidir autonomamente o que fazer; perseguir sua própria agenda; agir ao longo do tempo.

Como lembra Wooldridge, um sistema especialista é “desencorpado” e portanto, por natureza, não pode estar situado em um ambiente. Além disto, um sistema especialista não tem agenda própria, não é equipado com habilidades sociais, não age e não é capaz de ter um comportamento nem reativo nem pró-ativo.

Um robô por outro lado é claramente uma entidade situada em um ambiente. O robô certamente age, em geral persegue sua própria agenda e pode ser autônomo. Desta forma é relativamente claro que o paradigma de agente está relacionado muito mais à robótica do que aos sistemas especialistas.

A partir das definições de agentes vistas, pode-se analisar quais sub-áreas da IA clássica estão ou não de acordo com o paradigma de agentes e sistemas multiagentes. Isto fica claro no livro de Russell e Norvig onde em alguns capítulos a palavra agente não é sequer mencionada ou na melhor das hipóteses pode ser associada com um *componente* interno de um agente como poderia ser por exemplo um sistema especialista *embutido* na arquitetura de um agente.

3.12. Para saber mais sobre agentes autônomos e sistemas multiagentes

A Parte II deste capítulo introduziu de forma breve e seletiva apenas uma pequena parte do ferramental, das técnicas e dos conceitos que embasam a pesquisa tradicional e atual na área de agentes autônomos e sistemas multiagente. Esta seleção foi feita em função dos tópicos a serem abordados na Parte III. Desta forma, o leitor é remetido aos três livros que trazem este material de uma forma mais completa: [Weiss 1999], [Wooldridge 2009][9], e [Shoham and Leyton-Brown 2009].

Em relação aos tópicos abordados neste capítulo, estes podem ser aprofundados da seguinte forma. Cooperação e coordenação são tratados no capítulo 8 de [Wooldridge 2009]. Abordagens baseadas em teoria de jogos aparecem nos capítulos 3–6 de [Shoham and Leyton-Brown 2009] e nos capítulos 11 e 13 de [Wooldridge 2009]. Tópicos relacionados a este assunto, como leilões, votação, protocolos de negociação e alocação de tarefas são objeto dos capítulos 12 e 14 de [Wooldridge 2009], 9 e 11 de [Shoham and Leyton-Brown 2009], e 5 de [Weiss 1999]. Em particular, o tema projeto de mecanismos (*mechanism design*) que vem crescendo em importância, é bem abordado no capítulo 10 de [Shoham and Leyton-Brown 2009], aparecendo também no capítulo 14 de [Wooldridge 2009]. Aprendizado é o tópico dos capítulos 6 de [Weiss 1999] e 7 de [Shoham and Leyton-Brown 2009]. Comunicação é discutida no capítulo 7 de [Wooldridge 2009] e em [Shoham and Leyton-Brown 2009] no capítulo 8.

Para os conceitos básicos em torno de DPS e IAD, textos clássicos são: [Bond and Gasser 1988, Gasser and Huhns 1990, Jennings 1996].

Parte III: IA Multiagente – Problemas e Soluções

Os itens abordados na Parte I (Seções 3.3 a 3.6) são retrabalhados a seguir sob uma óptica multiagente, a partir dos conceitos introduzidos na Parte II.

3.13. Lógicas para sistemas multiagente

Em IA, um tópico ainda em discussão é sobre a adequabilidade do uso de lógica como um formalismo para representação do conhecimento. Atualmente é razoavelmente aceita a tese de que a resposta é sim, ou seja, formalismos lógicos satisfazem os requisitos básicos para representação do conhecimento, pelo menos no que se refere à sintaxe. Entretanto, alguns aspectos são tratados de forma pobre (ou não são tratados), especialmente quando se trata da lógica de primeira ordem.

[9] A numeração dos capítulos se refere à segunda edição.

A representação de conhecimento via lógica de primeira ordem (LPO) vista na Seção 3.3 não trata de forma simples questões fundamentais em sistemas de mais de um agente como por exemplo a questão dos estados mentais dos agentes. Isto entretanto não chega a ser um problema uma vez que existem outros formalismos lógicos para lidar com os estados mentais, os quais serão introduzidos a seguir.

Estados mentais – crenças, desejos, intenções – não são traduzíveis de forma simples utilizando-se a LPO. Por exemplo a sentença "Janine believes Cronos is the father of Zeus" não tem uma representação óbvia na LPO. Se a formalizarmos como "Bel (Janine, Father(Zeus,Cronos))", tal fórmula não é bem formada na LPO. Além disto, existe um problema semântico pois na LPO, o valor semântico da fórmula depende apenas dos valores-verdade dos componentes da fórmula (por exemplo ($p \vee q$) tem valor-verdade V ou F dependendo dos valores de p e de q). A fórmula "Janine believes p" tem um valor verdade que não depende apenas do valor de p. Igualmente, a fórmula "o agente_a acredita que o agente_b acredita que a lua é quadrada" não tem um valor verdade óbvio e, ainda que tivesse, isto nada nós diria sobre a lua ser ou não quadrada!

Desta forma, a LPO não é adequada para problemas que envolvem estados mentais e vários agentes. Isto é sabido desde pelo menos a publicação do trabalho de Hintikka em 1962 que introduziu a idéia de mundos possíveis ou seja uma "semântica" que depende do mundo no qual o agente vive. A formalização em lógica modal é devida a Kripke em 1963. Entretanto, os sistemas multiagentes que surgiram na década de 80 foram os que de fato levaram o foco da discussão para modelos pragmáticos que resultaram na chamada lógica BDI (de beliefs, desire, intention).

Lógica modal tem sido usada em IA para se referir a outras formas de dar significado à fórmulas (que não os modos simples da lógica de primeira ordem). Em filosofia, lógica modal é usada para se estudar outras formas de verdade como por exemplo possibilidade e necessidade. Em sistemas multiagentes ela é usada para raciocinar sobre crenças e conhecimento.

Para ilustrar este raciocínio, considere um agente jogando cartas (como pôquer), que tem um ás na mão (e sabe disto). Todas as configurações de mundo (alternativas epistêmicas) nas quais este agente não tem um ás não são possíveis. Pode-se inferir que um agente sabe p se p é verdadeiro em todas as alternativas epistêmicas do agente (por exemplo que tem um ás).

3.13.1. Lógica modal normal

A lógica proposicional multi-modal $\mathbf{K_n}$ estende a lógica proposicional por meio de n pares de operadores unários que são chamados operadores *box* ($\Box$) e *diamond* ($\Diamond$). $\mathbf{K}$ é dita multi-modal porque se consideram mais de um daqueles operadores.

Dependendo da aplicação desejada, tais operadores têm significados diferentes. Por exemplo, na lógica temporal usa-se o operador $\Diamond$ para se formalizar

o fato de que "em alguns momentos (no futuro) é o caso que ϕ". Isto é expresso da seguinte forma: $\Diamond \phi$. Os operadores $\Box$ e $\Diamond$ também podem significar necessariamente e possivelmente, respectivamente. Na lógica epistêmica, $\Box \phi$ é interpretado como “é sabido que ϕ”.

3.13.1.1. Sintaxe

O conjunto de fórmulas na lógica modal com símbolos proposicionais (proposições atômicas) p é o menor conjunto $\mathcal{L}$ que contem p tal que se $\phi, \psi \in \mathcal{L}$ então $\neg\phi \in \mathcal{L}$, $(\phi \wedge \psi) \in \mathcal{L}$ e $\Box\phi \in \mathcal{L}$. Assim como na lógica de primeira ordem, os demais conectivos ($\vee$, $\rightarrow$ e $\leftrightarrow$) podem ser derivados em termos de $\wedge$ e $\neg$. Adicionalmente pode-se derivar $\Diamond$ de $\Box$: $\Diamond\phi \equiv \neg\Box\neg\phi$ (ϕ é possivelmente verdadeiro se e somente se não ϕ não é necessariamente verdadeiro).

3.13.1.2. Semântica

A semântica de $\mathbf{K_n}$ é definida a partir do conceito de mundos possíveis ou estrutura de Kripke, $\mathcal{K} = (\mathcal{W}, \mathcal{R})$ que consiste em um conjunto $\mathcal{W}$ de mundos possíveis e um conjunto $\mathcal{R}$ de relações binárias de transição denominadas relações de acessibilidade. Cada mundo possível $w \in \mathcal{W}$ corresponde a uma interpretação ou seja uma atribuição de valor verdade $p^w \in \{0, 1\}$ para cada fórmula atômica p.

O conjunto $\mathcal{R}$ contem relações de transição $\mathcal{R}_m \subseteq \mathcal{W} \times \mathcal{W}$. Em uma estrutura de Kripke $\mathcal{K}$, a validade de uma fórmula ϕ em $\mathbf{K_n}$ no mundo $w \in \mathcal{W}$ é definida recursivamente:

$\mathcal{K}, w \vDash p$ sss $p^w = 1$ (para proposições atômicas p)
$\mathcal{K}, w \vDash \phi \wedge \psi$ sss $\mathcal{K}, w \vDash \phi$ e $\mathcal{K}, w \vDash \psi$
$\mathcal{K}, w \vDash \neg\phi$ sss não é o caso que $\mathcal{K}, w \vDash \phi$
$\mathcal{K}, w \vDash \Box\phi$ sss $\forall w' \in \mathcal{W}$ tal que $\mathcal{R}(w, w')$, é o caso de que $\mathcal{K}, w \vDash \phi$
$\mathcal{K}, w \vDash \Diamond\phi$ sss $\exists w' \in \mathcal{W}$ tal que $\mathcal{R}(w, w')$, é o caso de que $\mathcal{K}, w \vDash \phi$.

Uma fórmula ϕ de $\mathbf{K_n}$ é válida se e somente se é o caso que $\mathcal{K}, w \vDash \phi$ para todas estruturas de Kripke e todos os mundos w em $\mathcal{W}$.

3.13.1.3. Axiomatização

A lógica modal $\mathbf{K_n}$ não é suficiente para se atribuir um significado específico a um operador modal (por exemplo significados temporais como “no futuro” ou significados ligados à crença de um agente como "o robô acredita que ..."). De posse de noções semânticas de validade, é possível verificar se existe um sistema axiomático que permite derivar precisamente todas as fórmulas válidas. Serão introduzidos abaixo os axiomas do sistema axiomático $\mathbf{K}$ e as

regras de inferência correspondentes (*modus ponens* e necessitação):

Axioma **Taut**: todas as tautologias proposicionais são válidas
Axioma $\mathbf{K}$: $\Box\phi \wedge \Box(\phi \rightarrow \psi) \rightarrow \Box\psi$ é válida
Axioma $\mathbf{T}$: $\Box\phi \rightarrow \phi$
Axioma $\mathbf{4}$: $\Box\phi \rightarrow \Box\Box\phi$
Axioma $\mathbf{5}$: $\neg\Box\phi \rightarrow \Box\neg\Box\phi$ (ou $\Diamond\phi \rightarrow \Box\Diamond\phi$).
Modus ponens: de $\phi \rightarrow \psi$ e ϕ, inferir ψ (ou ainda: se ϕ e ($\phi \rightarrow \psi$) são válidas, inferir a validade de ψ)
Necessitação: da validade de ϕ, inferir a validade de $\Box\phi$.

Foi provado que o sistema axiomático $\mathbf{K}$ é correto e completo para toda classe de modelos de Kripke.

A lógica modal $\mathbf{K_n}$ é caracterizada pelos axiomas **Taut** e $\mathbf{K}$; uma fórmula ϕ de $\mathbf{K_n}$ é válida em todos os mundos de todas as estruturas de Kripke se e somente se ela puder ser derivada de instâncias de **Taut** e $\mathbf{K}$ usando as regras de inferência *modus ponens* e necessitação.

Na lógica de conhecimento discutida na próxima seção, os três demais axiomas descrevem possíveis propriedades dos operadores modais relacionados ao conhecimento de um agente. Por exemplo $\Box\phi$ deve ser entendida como "o agente sabe ϕ". Logo, o axioma $\mathbf{T}$ deve ser entendido como “se um agente sabe ϕ em uma situação, então ϕ é válida nesta situação” o que implica que um agente não tem conhecimento incorreto. Notar entretanto que há uma diferença entre conhecimento e crença: um agente pode acreditar em fatos incorretos mas ele não tem como saber isto.

O axioma $\mathbf{4}$ ($\Box\phi \rightarrow \Box\Box\phi$ ou introspeção positiva) pode ser entendido como "o agente sabe o que ele sabe". Já o axioma $\mathbf{5}$ ($\neg\Box\phi \rightarrow \Box\neg\Box\phi$ ou introspeção negativa) indica que "o agente sabe o que ele não sabe", o que é questionável (você sabe tudo o que não sabe?). Desta forma, alguns sistemas axiomáticos dispensam o axioma $\mathbf{5}$ pois ele implica que um agente sabe que seu conhecimento não é suficiente para resolver certas tarefas. Desta forma, o chamado sistema $\mathbf{S4}$ não inclui a introspeção negativa.

Existem outras propriedades associadas ao conhecimento que não são deriváveis do sistema axiomático $\mathbf{K}$. Uma delas é a consistência (do conhecimento) que se refere ao fato de que o conhecimento de um agente não é contraditório. O axioma $\mathbf{D}$ captura exatamente esta ideia:

Axioma $\mathbf{D}$: $\neg\mathbf{K}_i(\phi \wedge \neg\phi)$ ou $\mathbf{K}_i\phi \Leftrightarrow \neg\mathbf{K}_i\neg\phi$)

3.13.2. Lógica de conhecimento para sistemas multiagentes

A lógica modal que trata de conhecimento é a lógica epistêmica. Conforme mencionado, nesta lógica $\Box\phi$ deve ser entendida como "é sabido que ϕ". Entretanto, desta forma, esta lógica lida com o conhecimento de um único agente.

Para lidar com conhecimento multiagente é necessário adicionar um conjunto de relações de acessibilidade, uma para cada agente. O modelo estendido é $\mathcal{K} = (\mathcal{W}, \mathcal{R}_1, ..., \mathcal{R}_n)$ onde cada $\mathcal{R}_i$ se refere às relações de acessibilidade do conhecimento do agente i. Desta forma, o operador modal $\Box$ introduzido anteriormente é substituído por um conjunto de operadores modais unários $\mathbf{K}_i$ onde i denota um agente. Cada operador $\mathbf{K}_i$ tem as mesmas propriedades do operador $\Box$. Podemos, a partir dai, nos perguntar como $\mathbf{K}_n$ é útil como lógica de conhecimento (e também de crença).

A regra de necessitação nos diz que "um agente sabe todas as fórmulas válidas", incluindo todas as tautologias, cujo número é infinito. Disto decorre que um agente sabe um número infinito de itens de conhecimento, o que parece ser um propriedade muito forte de agentes que têm limitações (de recursos, etc.).

Além da regra de necessitação, quão adequados são os demais axiomas? Já analisamos os axiomas $\mathbf{T}$, $\mathbf{4}$ e $\mathbf{5}$ quando aplicados à lógica de conhecimento quando discutimos introspeção positiva e negativa. Resta o axioma $\mathbf{K}$ que relaciona o conhecimento de um agente com um fecho de implicação. Suponha que ϕ é consequência lógica do conjunto $\Phi = (\phi_1, ..., \phi_n)$. Em cada mundo onde todas ϕ_i são verdadeiras, ϕ também será e portanto $\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_n \Rightarrow \phi$ deve ser válida. Pela regra de necessitação, da validade de ϕ infere-se $\Box\phi$ (ϕ é conhecida pelo agente). Isto é conhecido como ominiscência lógica. Por exemplo, se um agente sabe todos os axiomas de Peano, então ele sabe todos os teoremas da aritmética, o que, novamente, é uma propriedade muito forte e não se espera que um agente limitado tenha tal propriedade.

3.13.3. Exemplos do uso do formalismo de Kripke em sistemas multiagentes

O seguinte exemplo visa ilustrar os problemas relacionados à presença de vários agentes com conhecimento parcial. Trata-se do problema do ataque coordenado.

Dois generais A e B e seus exércitos estão posicionados no topo de duas colinas. Eles precisam se comunicar a fim de coordenar um ataque a um exército inimigo, que se encontra no vale. A comunicação se dá através de mensageiros (os generais não possuem meios mais sofisticados de comunicação!), que para ir de A a B devem passar pelo vale onde se encontra o inimigo C. Caso um mensageiro seja capturado, a mensagem é perdida. A e B só têm chance de sucesso se o ataque for simultâneo; se apenas A ou B atacam, C vence.

Dada tal situação, qual protocolo A e B devem estabelecer a fim de garantir um ataque simultâneo? Um protocolo simples mas ingênuo seria o seguinte: B envia a A, a cada hora, uma mensagem "ataque amanhã ao amanhecer", até que receba um ack (confirmação de recebimento) de A. O mesmo protocolo preve que A não faça nada até receber uma mensagem de B, a qual deve ser respondida com um ack para B.

Dado este protocolo, é garantido que o ataque simultâneo ocorrerá? Se não, qual seria uma alternativa? Neste exemplo ilustrativo, nota-se claramente que as noções de conhecimento discutidas na Seção 3.13.2 têm um papel importante mas como podemos utiliza-las para formalizar este cenário? Além disto, que conhecimento é suficiente para garantir o ataque coordenado?

Pensando na noção de mundos possíveis, podemos definir os estados (de conhecimento) locais e globais (combinados). Seja um estado um par de variáveis binárias que refletem se uma mensagem foi enviada ou não (primeira variável) e se um ack foi enviado ou não (segunda variável). Desta forma, o par $(0,0)$ indica que a mensagem não foi enviada e que o ack não foi enviado. Existem portanto 4 estados locais possíveis para cada general A e B. O número de estados globais é 16 pois para cada estado local de A, B pode estar em 4 estados.

Suponhamos que o estado global inicial é $s^0_{AB} = \langle(0,0),(0,0)\rangle$. As transições de estado são determinadas pelo protocolo (determinístico) e pela natureza ou ambiente, este não previsível pois por exemplo não podemos prever se o mensageiro será ou não capturado. Assim, a partir de s^0_{AB} se sucedem novos estados $s^1_{AB}, s^2_{AB}, ...$ O conjunto destes estados denominaremos de histórico e eles constituem os mundos possíveis do nosso problema.

Nestes mundos possíveis, o conhecimento pode ser expresso da seguinte forma. Seja $p = attack$ a fórmula atômica associada ao fato de que um ataque é planejado em determinada hora. $\mathbf{K}_i\ p$ significa que i sabe que deve atacar. Por exemplo, em um mundo possível cujo último estado global registrado seja $\langle(1,1),(1,1)\rangle$, as seguintes fórmulas são válidas: $\mathbf{K}_B\ p$, $\mathbf{K}_A\ p$, $\mathbf{K}_B\mathbf{K}_A\ p$. Entretanto, a fórmula $\neg\mathbf{K}_A\mathbf{K}_B\mathbf{K}_A\ p$ também é válida pois A não sabe que B sabe que A sabe que B ordenou o ataque, já que A sabe que o último ack enviado pode ter sido perdido. Logo, A não ataca.

É fácil perceber que o protocolo não garante o ataque simultâneo. Como o protocolo pode ser melhorado? É preciso adicionar a noção de conhecimento comum (common knowledge). Para isto é preciso primeiro definir a noção de *everybody knows* (todo mundo sabe), representada pelo operador modal E_G onde G é um grupo de agentes (doravante este subscrito será omitido sempre que não causar ambiguidade, subentendendo-se que o grupo inclui todos os agentes). Assim, $E\phi$ significa que cada agente sabe ϕ.

Formalmente, E é definido da seguinte maneira. Seja $\mathcal{K}$ uma estrutura de Kripke, $w \in \mathcal{W}$ um mundo possível, G um grupo de agentes e ϕ uma fórmula da lógica modal. $\mathcal{K}, w \vDash E_G\phi$ sss para todo $i \in G$ é o caso que $\mathcal{K}, w \vDash \mathbf{K}_i\phi$. Isto significa que todos os agentes sabem ϕ quando ϕ for verdadeira em todos os mundos que são possíveis.

A noção de conhecimento comum entretanto não é capturada pelo operador E. É necessário definir um outro operador, C, que é definido em função de E. Sejam $\mathcal{K}, G, \phi$ como definidos acima. $\mathcal{K}, w \vDash C_G\ \phi$ sss $\mathcal{K}, w \vDash \mathbf{E}_G(\phi \wedge C_G\ \phi)$. Em outras palavras, ϕ é conhecimento comum se e somente se todo mundo sabe ϕ e todo mundo sabe que ϕ é conhecimento comum. Esta definição não

é trivial já que $C_G\ \phi$ é o equivalente à solução de ponto fixo de uma equação. Existe entretanto uma definição semântica equivalente: $\mathcal{K}, w \vDash C_G\ \phi$ sss $\mathcal{K}, w' \vDash \phi$ para todas as sequências de mundos possíveis $w = w_0, w_1, ..., w_n = w'$ para as quais o seguinte vale: para cada $0 \leq i < n$ existe um agente $j \in G$ tal que $w_{i+1} \in I_j(w_i)$, onde $I_j(w) = \{w' | w \in \mathcal{W}_{j_k}\}$ e $w' \in \mathcal{W}_{j_k}$ ou seja $I_j(w)$ inclui todos os mundos na partição w de acordo com o agente j.

3.13.4. Combinando conhecimento e crença

Tal como o conhecimento, as crenças são atitudes mentais, estas relacionadas à visão do agente. Desta forma, pode-se utilizar as estruturas de Kripke (mundos possíveis e relações entre eles) para modelar crenças, ainda que modificações sejam necessárias aqui.

Retornando à noção de sistema axiomático introduzida na Seção 3.13.1.3 e posteriormente discutida na Seção 3.13.2 para o caso particular de conhecimento, lembramos que o axioma $\mathbf{T}$ ($\mathbf{K}_i, \phi \rightarrow \phi$) faz parte do sistema axiomático lá visto. Assumamos que $\mathbf{T}$ não faz parte daquele sistema. Desta forma, o sistema axiomático visto na Seção 3.13.2 pode agora ser revisitado a fim de definí-lo para o operador modal $\mathbf{B}$ (de belief, crença) no lugar de $\mathbf{K}$:

Axioma $\mathbf{K}$: $\mathbf{B}_i\phi \wedge \mathbf{B}_i(\phi \rightarrow \psi) \rightarrow \mathbf{B}_i\psi$ é válida
Axioma $\mathbf{D}$: $\neg\mathbf{B}_i(p \wedge \neg p)$
Axioma $\mathbf{4}$: $\mathbf{B}_i\phi \rightarrow \mathbf{B}_i\mathbf{B}_i\phi$
Axioma $\mathbf{5}$: $\neg\mathbf{B}_i\phi \rightarrow \mathbf{B}_i\neg\mathbf{B}_i\phi$.

As duas regras de inferência têm o mesmo papel já definido anteriormente.

Conhecimento e crença podem ser combinados a fim de se poder expressar por exemplo que “se Bob sabe que Alice acredita que está chovendo, então Alice sabe que Bob sabe isto” (o que pode ser formalizado como: $\mathbf{K}_A\mathbf{B}_A rain \rightarrow \mathbf{K}_A\mathbf{K}_B\mathbf{B}_A rain$).

3.13.5. Objetivos e desejos

Em última análise, o que se deseja é que agentes possam modelar objetivos e desejos, além de conhecimento e crenças. Afinal, na nossa definição de agente, a capacidade de agir para atingir um objetivo é uma das noções fundamentais.

Cohen e Levesque [Cohen and Levesque 1990] adaptaram a semântica de mundos possíveis a fim de desenvolver uma lógica de objetivos, intenções e desejos. Nesta adaptação, cada mundo acessível, para cada objetivo em particular, representa uma instância na qual o mundo pode passar a estar, caso o objetivo do agente seja realizado. Um dos problemas com esta abordagem é o chamado efeito colateral que é associado ao fato de que um agente terá como objetivo todas as consequências lógicas do objetivo inicial. Um exemplo que ilustra este efeito é o seguinte; um agente tem como objetivo ir ao dentista (cuja consequência é sentir dor) mas é óbvio que sentir dor não é um objetivo do agente. Uma adaptação no modelo de mundos possíveis foi proposta em

[Wainer 1994] para resolver este problema.

A despeito de alguns problemas remanescentes, a lógica de intenções de Cohen e Levesque representa um passo significativo na direção de uma teoria de agência. Esta lógica tem tido grande utilidade nas áreas de análise de conflito e cooperação entre agentes tendo sido posteriormente estendida para tratar intenções conjuntas (*joint intentions*).

A sintaxe da lógica de intenções considera quatro operadores: $Bel_i\ \phi$ (agente i acredita ϕ); $Goal_i\ \phi$ (i tem como objetivo ϕ); $Happens\ \alpha$ (ação α ocorrerá em seguida); $Done\ \alpha$ (ação α foi executada). Outros construtores advindos de outros formalismos para tratar ambientes dinâmicos foram posteriormente introduzidos como $\alpha; \alpha'$ (α ocorre após α').

3.14. Resolução de problemas em sistemas multiagentes

Conforme visto na Seção 3.4, o formalismo em torno de CSP é útil para certos problemas de planejamento e busca quando colocados de forma *centralizada*. Entretanto, no mundo real existem situações nas quais o problema necessita ser resolvido de forma distribuída, seja porque não há capacidade de processamento central, seja porque os recursos e o conhecimento se encontram de fato distribuídos, ou ainda devido a altos custos de comunicação para centralização de todo o problema em um único local. Além disto podem haver questões intrínsecas de segurança e privacidade das informações. Um exemplo típico é o de agendamento de reuniões. Neste problema, em geral, as restrições bem como as informações sobre horários não são de domínio público e/ou existem questões de privacidade que implicam que nenhuma entidade central tem todas as informações a fim de resolver o problema usando CSP clássico.

Para especificar um CSP distribuído, foi proposto o chamado DisCSP (do inglês, *distributed constraint satisfaction problem*) [Yokoo et al. 1992]. Já o formalismo que especifica COPs de maneira distribuída é denominado DCOP (do inglês *distributed constraint optimization problem*) [Modi et al. 2003]. Em ambos os formalismos que tratam de problemas em ambientes distribuídos, utilizam-se agentes para representar e controlar as variáveis.

Tanto em DisCSP quanto em DCOP, cada variável é gerenciada por um agente. O objetivo global continua sendo o mesmo. Entretanto, cada agente tem agora autonomia para decidir o valor que será atribuído à variável. Este problema está longe de ser trivial já que nenhum agente tem uma visão global. Desta maneira, cada agente tem que se comunicar com outros agentes – aqueles que fazem parte da sua vizinhança no grafo de restrições.

3.14.1. Problema de satisfação de restrições distribuídas

Um DisCSP é um CSP em que as variáveis e restrições estão distribuídas entre agentes autônomos, constituindo um sistema multiagente. Cada agente é responsável por uma variável e deve determinar o seu valor a partir dos valores disponíveis no domínio da variável. Como existem restrições entre variáveis, existem interações entre agentes (na forma de restrições). Logo, a atribuição

que constitui a solução para o DisCSP deve satisfazer estas restrições entre agentes.

Formalmente, em um DisCSP temos um conjunto $A = \{a_1, \ldots, a_m\}$ de m agentes. Cada variável x_j pertence a um agente a_l. Esta relação é representada pelo predicado $Pertence(x_j, a_l)$. As restrições também estão distribuídas entre os agentes. O fato de que um agente a_l conhece uma restrição C_k que atua sobre a variável do agente é denotado pelo predicado $Conhece(C_k, a_l)$. A solução de um DisCSP é uma atribuição completa $\mathcal{A}$ que satisfaz as seguintes condições:

- $\forall a_l, \forall x_j$ onde $Pertence(x_j, a_l)$, o valor atribuído a x_j é d_j e;
- $\forall a_l, \forall C_k$ onde $Conhece(C_k, a_l)$, C_k é satisfeita com a atribuição $\langle x_j, d_j \rangle$.

Assim como um CSP, um DisCSP também pode ser representado por um grafo de restrições. Neste caso, os vértices são os agentes e as arestas são as restrições entre os agentes.

O algoritmo fundamental para resolver um DisCSP é o ABT (asynchronous backtracking). O ABT assume uma ordenação total dos agentes. Cada restrição binária é conhecida por ambos os agentes e o agente com menor prioridade é responsável por verificar a restrição após ter recebido a mensagem sobre a atribuição do(s) agente(s) de maior prioridade. Uma aresta é direcionada do agente de maior para o de menor prioridade.

As atribuições de valores são feitas em paralelo, respeitando as restrições conhecidas em cada agente. Os valores são comunicados para os agentes vizinhos (aqueles com os quais existe uma aresta em comum). No passo seguinte, todas mensagens são processadas e respondidas. A seguir, uma nova atribuição de valores é feita, respeitando as já conhecidas, e novas mensagens são enviadas.

Mensagens do tipo **OK?** são as que informam uma atribuição realizada. Quando o agente a_i recebe uma mensagem **OK?** do agente a_j, a_i coloca a atribuição agora conhecida em uma estrutura de dados denominada **agent_view**. Após, a_i verifica se sua atribuição atual é consistente com sua **agent_view**. Se não for, a_i tenta atribuir outro valor dentro do domínio possível. Se não encontrar, a_i inicia um processo de backtracking enviando uma mensagem **nogood** a a_j. No ABT a **nogood** é composta de toda a **agent_view**.

O ABT é a base de várias extensões propostas. Uma delas é o envio apenas do conjunto mínimo que justifique uma inconsistência. Dado que encontrar este conjunto mínimo é um problema NP-difícil, encontrar métodos heurísticos que não sacrifiquem a corretude é um desafio.

O algoritmo *asynchronous weak-commitment search* é uma modificação do ABT na qual as variáveis iniciam com valores-tentativos. Uma solução parcial consistente é construída para subconjuntos de variáveis. Cada solução parcial é estendida com a adição de variáveis (e seus valores), uma a uma, até que uma solução completa seja encontrada. Este algoritmo é capaz de revisar uma atribuição ineficiente sem a necessidade de uma busca exaustiva.

3.14.2. Problema de otimização de restrições distribuídas

Conforme mencionado na Seção 3.4, há casos nos quais deseja-se encontrar a *melhor* e não qualquer solução para um problema. A versão distribuída deste caso é denominada de DCOP. Em DCOP's valem as mesmas hipóteses e técnicas básicas de um DisCSP mas aqui é necessário que os agentes encontrem, de forma colaborativa, a melhor solução para o problema.

Um DCOP estende um DisCSP com a inclusão de uma função objetivo $\mathcal{F}(\mathcal{A})$ e funções de custo f (como acontece com um COP em relação a um CSP). O objetivo é novamente encontrar uma atribuição $\mathcal{A}^*$ que proporcione o valor ótimo para a função objetivo. Na Figura 3.4 temos um exemplo de DCOP na forma de grafo de restrições, com quatro agentes (variáveis) $\{x_1, x_2, x_3, x_4\}$ e quatro funções de custo (arestas) $\{f(x_1, x_2), f(x_1, x_3), f(x_2, x_3), f(x_2, x_4)\}$. Trata-se de um problema onde cada variável pode assumir apenas 2 valores: $\{0, 1\}$. A função objetivo para o DCOP da Figura 3.4 corresponde a agregação das funções de custo:

$$\mathcal{F}(\mathcal{A}) = \sum_{x_i, x_j \in X} f_{ij}(d_i, d_j), \text{ onde } x_i \leftarrow d_i, x_j \leftarrow d_j$$

A solução deste DCOP exemplo é a atribuição completa
$\mathcal{A}^* = \{\langle x_1 = 1\rangle, \langle x_2 = 1\rangle, \langle x_3 = 1\rangle, \langle x_4 = 1\rangle\}$.

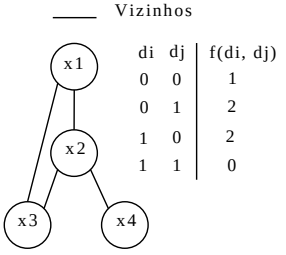

Figura 3.4. Exemplo de DCOP

Vários algoritmos completos[10] têm sido propostos para resolver DCOP's; este texto apresenta brevemente os dois mais populares: Adopt e DPOP. Ambos requerem que os agentes estejam organizados de alguma forma para estabelecer uma ordem de prioridade entre os mesmos. Esta ordem de prioridade tem dois propósitos: 1) formar uma estrutura de comunicação acíclica entre os agentes (evitando ciclos infinitos no fluxo de comunicação que comprometeriam a completude dos algoritmos); 2) definir os destinatários dos diferentes

[10] Devido à complexidade do problema quando aplicado a cenários com muitos agentes /ou ligados à alocação de tarefas, foram propostos também algoritmos aproximados como LA-DCOP e Swarm-GAP, porém estes não são abordados aqui.

tipos de mensagens utilizadas pelos algoritmos na comunicação entre os agentes.

A maneira mais simples de estabelecer esta prioridade entre os agentes é através de uma ordenação linear (total) a partir do grafo de restrições. Esta ordenação pode ser obtida, por exemplo, através do identificador do agente. Agentes com menor identificador têm maior prioridade, enquanto que agentes com maior identificador têm menor prioridade. A outra forma é organizar os agentes em uma árvore de busca binária; esta é a forma utilizada pelo Adopt e DPOP.

O Adopt (*Asynchronous Distributed Optimization*) [Modi et al. 2003] é um algoritmo totalmente assíncrono para resolver DCOP's, que é fundamentado na técnica de *backtracking* para realização da busca pelo custo ótimo. Devido ao uso desta técnica, a complexidade de espaço do algoritmo é linear. O *backtracking* é feito utilizando uma estratégia *melhor primeiro*. Nesta estratégia, cada agente escolhe o valor de domínio que apresenta a melhor estimativa de custo ótimo, dadas as informações locais que o agente tem sobre os demais.

No Adopt, esta estimativa de custo ótimo é denominada de limitante inferior (*lower bound*) e deve, necessariamente, subestimar o custo ótimo. Em função do limitante inferior ser uma estimativa estabelecida com base em informações locais, torna-se muito mais vantajosa a sua utilização em algoritmos assíncronos, pois não há a necessidade de aguardar (síncronamente) por uma grande quantidade de informação a respeito de outros agentes para computar o custo. Pode-se, por exemplo, estabelecer um limitante inferior imediatamente a partir das funções de custo conhecidas pelo agente. Esta é uma grande vantagem sobre, por exemplo, algoritmos do tipo *branch and bound*, pois estes requerem que seja estabelecido um limitante superior global, que leve em consideração informações sobre todos os agentes, para servir de corte no processo de busca.

Além do limitante inferior, cada agente também mantém um *limitante superior* (*upper bound*). Este limitante superior constitui o efetivo custo (e não uma estimativa) de uma solução parcial, dadas as informações disponíveis ao agente. Além de ser útil para podas de valores de domínio que geram custos superiores, o limitante superior também serve para estabelecer a condição de término do algoritmo: quando nenhum valor de domínio proporciona custo menor que este limitante.

O limitante inferior de cada agente no Adopt é refinado iterativamente a partir de novas informações de custo recebidas assincronamente de outros agentes. Este refinamento, combinado com a estratégia oportunista de escolher sempre o valor com melhor estimativa, pode levar ao fato de que um valor de domínio seja descartado antes de ser efetivamente comprovado que ele não constitui a solução ótima. Se isto ocorrer, é possível que este valor venha a ser revisitado, em algum momento futuro da busca.

A comunicação no Adopt é realizada a partir de quatro tipos de mensagens: VALUE é enviada de pai para filhos e pseudo-filhos e contém a atribuição com

o valor atual do agente. Seu objetivo é informar aos filhos/pseudo-filhos o valor da variável, para que estes possam computar o custo em função deste valor. A mensagem THRESHOLD é enviada de pai para filho e corresponde ao valor de *threshold* que o filho deve considerar.

A mensagem COST é enviada de filho para pai com o objetivo de informar ao pai qual é o *limitante inferior* e *superior* que foram computados localmente após ser conhecido o valor do agente pai. Com esta informação de custos, o agente pai pode decidir se estes custos são satisfatórios ou se deve procurar por um custo menor através da escolha de outro valor do domínio.

Por fim, a mensagem TERMINATE é enviada de pai para filhos/pseudo-filhos e indica que o agente pai encontrou a solução ótima e está finalizando a computação para que os filhos saibam que o valor do pai não apresenta alterações ou seja tornou-se estável; com isto os filhos também atingem a condição de término.

Sendo baseado em *backtracking*, o Adopt tem como principal vantagem o fato de ser linear em termos de memória e no *tamanho* das mensagens trocadas, ainda que seja exponencial em termos de tempo. Entretanto, sempre que um agente altera seu valor, ao menos uma mensagem é necessária para informar aos demais agentes esta alteração. Os demais agentes, ao receberem esta mensagem, atualizam o respectivo contexto para refletir o novo valor do agente remetente. Esta necessidade de ao menos uma mensagem para cada alteração de valor pode tornar exponencial o *número* de mensagens requeridas pelo algoritmo para computar a solução.

Para contornar o problema do excessivo número de mensagens, foi proposto um algoritmo baseado na técnica de programação dinâmica, denominado DPOP (*Distributed Pseudotree Optimization*) [Petcu and Faltings 2005]. DPOP usa uma árvore de busca em profundidade onde cada sub-árvore constitui um sub-problema independente com sub-estrutura ótima. Sendo assim, a solução ótima para uma sub-árvore faz parte da solução ótima do problema como um todo. Cada agente raiz de uma sub-árvore computa o custo ótimo levando em consideração todas as combinações de valores de domínio possíveis entre si e os agentes filhos e pais. Estes custos são propagados até o raiz da árvore, que tem então toda informação (de custos) necessária para escolher qual é a solução ótima. Para propagar estes custos é utilizado um vetor de utilidade.

O algoritmo DPOP possui três fases. A primeira compreende a criação da árvore; a segunda fase consiste da propagação dos vetores de utilidade; a terceira fase é a propagação dos valores selecionados por cada agente.

A propagação do vetor de utilidade é feita através de mensagens UTIL. Cada mensagem UTIL é enviada apenas ao pai. Um agente folha x_{folha} calcula o vetor de utilidade $UTIL_{x_{folha}}$ levando em consideração os seus valores de domínio e os valores de domínio dos agentes pseudo-pais e do agente pai. Cada agente intermediário x_i na árvore recebe os vetores de utilidade $UTIL_{x_{filho}}$ provenientes de cada um dos agentes filhos.

A terceira fase, propagação de valores escolhidos, consiste em cada agente

selecionar o valor com custo ótimo para si e enviar este valor para seus filhos através de mensagem VALUE. Cada folha, tendo agora conhecimento dos valores de todos os ascendentes envolvidos em restrições consigo, escolhe para si o valor com custo ótimo.

Por construção, o número de mensagens utilizadas em uma computação com o DPOP é linear. O número de ciclos requerido também é linear, e corresponde a duas vezes a profundidade da árvore, pois cada nível requer um ciclo para propagar o vetor de utilidade (mensagens UTIL) e outro para propagar os valores ótimos selecionados (mensagens VALUE). Em contrapartida, o tamanho dos vetores de utilidade (e conseqüentemente o tamanho das mensagens UTIL) é exponencial em função da quantidade de pseudo-pais que cada agente pode ter. Quanto maior a quantidade de pseudo-pais, mais dimensões haverá no vetor de utilidade. Como cada dimensão do vetor contem todos os valores de domínio do respectivo agente, temos também que quanto maiores os domínios, mais elementos haverá no vetor de utilidade.

3.15. Planejamento multiagente

O que acontece com o problema do planejamento descrito na Seção 3.5 se a tarefa de planejar depende de ou envolve outros agentes? Vários problemas são agora introduzidos. Cada agente pode ter seu próprio objetivo; cada agente tem níveis de privacidade que o impede de divulgar todas as informações; o conhecimento não é global; um planejamento clássico, centralizado, pode não ser viável (dadas questões de complexidade).

Tais fatos sugerem que se tenha que trabalhar com soluções parciais e com cada agente gerando seu próprio plano. O único problema é que estes planos muito provavelmente são interelacionados e interferem uns nos outros. Se eventuais conflitos (por exemplo uso de recursos comuns) não forem tratados, todos os planos podem ser inviabilizados por aspectos não previstos. Para resolver este conflitos, ações de coordenação entre os agentes são necessárias. Por isso se diz que:

Multi-agent planning = planning + coordination

Historicamente a área de planejamento multiagente está fortemente associada com DPS (Seção 3.7) pois trata de fazer com que agentes trabalhem juntos para resolver problemas. Contrariamente ao planejamento monoagente, no caso multiagente, recursos, capacidades, conhecimento, etc. encontram-se distribuídos entre os agentes de modo que nenhum é capaz de realizar uma tarefa sozinho.

Um exemplo clássico de distribuição de conhecimento é o seguinte. Suponha que em um grupo de 2 agentes, A sabe ϕ e B sabe $(\phi \rightarrow \psi)$. Há um conhecimento distribuído de ψ embora nem A nem B explicitamente saibam ψ.

Em [Durfee 1999] são listados vários cenários e métodos para DPS como *task sharing*, *result sharing* e planejamento. Nesta seção nos deteremos no último método. Em resumo, o problema clássico de planejamento definido anteriormente, na sua versão multiagente agora é estendido da seguinte forma.

Dados: i) descrição do estado inicial; ii) conjunto de objetivos globais; iii) conjunto de agentes; iv) para cada agente, um conjunto de habilidades e objetivos privados; a tarefa é encontrar um plano para cada agente que atinja seus objetivos privados mas que sejam coordenados de forma a garantir que o objetivo global seja atingido.

Nesta seção será discutido o que muda quando há mais de um agente agindo no ambiente e/ou realizando o planejamento de forma distribuída. Para o primeiro caso é possível se utilizar o formalismo já visto na Seção 3.5 sendo que cada agente simplesmente inclui os demais, estendendo seu modelo de ambiente. Com isto seria possível utilizar os algoritmos já vistos. Entretanto esta abordagem ingênua tende a falhar e/ou transformar-se em um problema de complexidade alta e/ou baixo desempenho, se for viável em primeiro lugar.

O segundo caso (planejamento completamente distribuído) será discutido adiante. Antes, serão abordados os casos nos quais algum tipo de centralização existe. Considere o caso no qual os objetivos e os planos são comuns como é o caso de agentes jogadores de futebol no ambiente da RoboCup Soccer. Neste caso, uma primeira extensão necessária aos formalismos já vistos é a de que é preciso explicitar qual agente realiza qual ação. Além disto é preciso que cada ação também esteja relacionada a um agente.

Uma primeira proposta para resolver o problema de planejamento multiagente é de alguma forma centralizada: no chamado plano conjunto centralizado, existe na verdade apenas uma instância de plano e esta preve que ações cada agente deve executar. Na realidade este formalismo não difere significativamente dos vistos na Seção 3.5, embora seja mais complexo pois uma entidade centralizadora deve realizar o planejamento considerando todos os agentes. Desta forma, este formalismo não será detalhado pois não nos atende, seja por não ser verdadeiramente multiagente, seja por ser muito complexo.

No chamado plano conjunto descentralizado, cada agente realiza o cálculo dos planos. Dado que o objetivo é comum e que possivelmente os agentes têm a mesma base de conhecimento sobre o estado inicial do ambiente, eles podem determinar o(s) plano(s) de ação. O problema ocorre no caso de haver mais de um plano possível e os agentes não coordenarem a escolha destes planos. No exemplo do futebol de robôs, assumindo-se apenas dois agentes atacantes (além do goleiro do time adversário), o plano conjunto dos dois atacantes certamente falhará se ambos escolherem um plano que preveja como ação inicial a espera da bola via passe pelo outro atacante.

Uma solução para este problema é a explicitação das ações concorrentes possíveis ou ações conjuntas (*joint actions*). A partir deste conjunto, um plano consiste de um grafo parcial ordenado de ações conjuntas. Para evitar a descrição de *todas* as ações conjuntas (imagine especificar este conjunto para um time de 11 jogadores, cada qual com um número de aproximadamente uma dezena de ações!), pode-se listar apenas as ações que efetivamente interagem (por exemplo as ações do goleiro pouco interagem com as dos atacantes

de seu time). Para um formalismo baseado em STRIPS, isto significa a adição de uma lista de ações conjuntas (à semelhança da lista de pré-condições), somente que esta especifica a lista de ações possíveis de serem executadas concomitantemente. No exemplo dos dois atacantes, tal ação seria:
$Action(wait(A, pass))$,
$CONCURRENT : \neg wait(B, pass)$,
$PRECOND : ...$
$EFFECT : ....$

Durfee vê planejamento distribuído como uma especialização de DPS onde o problema em si é formular um plano. Há várias formas de realizar isto conforme segue.

3.15.1. Planejamento centralizado para planos distribuídos

Nesta variante os planos são formulados de forma centralizada e executados de forma distribuída como é o caso de um planejamento de ordem parcial onde não há ordenação rígida entre algumas ações e que pode portanto ser executado em paralelo por vários agentes.

Um elemento centralizador é responsável por formular um plano, decompô-lo entre os agentes e incluir ações de sincronização (tipicamente atos comunicativos). O problema com esta abordagem é que ela assume conhecimento correto (por exemplo sobre a capacidade dos agentes) e total observação do mundo, além de não prever falhas na execução e de eventualmente resultar em um grande volume de comunicação.

3.15.2. Planejamento distribuído para planos centralizados

Nesta variante, formular um plano é visto como uma tarefa colaborativa entre vários especialistas, onde cada um contribui com uma parte do plano. Após, um elemento planejador ordena estas partes. Se alguma das partes não pode ser realizada (devido e.g. a restrições não satisfeitas), é possível se utilizar algum mecanismo de CSP distribuído (conforme Seção 3.14.1) para tentar corrigir o plano. Eventualmente não há solução e mecanismos de otimização de restrições (Seção 3.14.2) devem ser utilizados (a um custo).

A tarefa de planejamento distribuído pode ser feita de forma parcial com compartilhamento do plano (parcial) e posterior combinação deste (*merging*), sendo estas duas atividades repetidas até que haja convergência para um plano único, o que se constitui um problema típico de CSP distribuído.

3.15.3. Planejamento distribuído para planos distribuídos

Esta é a variante mais desafiadora pois tanto a tarefa de planejar quanto seu resultado são distribuídos. Paradoxalmente, aqui pode não ser necessário ter o plano representado em sua totalidade num único local. Entretanto, as partes componentes do plano precisam ser compatíveis. Os métodos reportados na literatura são diversos, sendo os principais apresentados a seguir.

Denomina-se *plan merging* o método no qual diversos agentes formulam planos individuais para seus próprios propósitos, os quais devem ser executa-

dos em paralelo mas sem conflitos (por exemplo quando tarefas tentam utilizar um mesmo recurso). Aqui o desafio é identificar e resolver conflitos em potencial. Para tanto, um método é ter um agente que colete todos os planos, analise-os, identifique e resolva os conflitos. Em geral tal método é impraticável se resolvido por enumeração de todos os estados finais que derivam dos estados iniciais e da realização de cada uma das ações possíveis em cada estado intermediário, no estilo de um MDP (Seção 3.5.2).

Para lidar com esta complexidade, [Georgeff 1983] propôs uma representação baseada em STRIPS (Seção 3.5), onde cada pré-condição deve valer para que uma ação possa ser considerada, reduzindo assim o espaço de possíveis ações.

Para o merging propriamente dito dos planos, o agente que coletou estes planos considera pares de ações como por exemplo a_i e b_j provenientes de planos propostos por diferentes agentes A e B. As ações a_i e b_j podem ser executadas em paralelo se suas pré-condições, condições e efeitos são satisfatíveis ao mesmo tempo. Neste caso, diz-se que as ações são independentes. Em caso contrário elas devem obedecer uma ordenação, ou seja não podem ser realizadas em paralelo pois determinadas restrições devem ser respeitadas. Neste caso, o problema pode ser formulado como um CSP ou um COP distribuído, este último remetendo à abordagens que tentam maximizar o desempenho global. Em qualquer dos casos, *plan merging* tem a desvantagem de se apoiar em um agente coletor o que dá um caráter centralizador.

Um segundo método para esta classe de problemas é o *iterative plan formulation*, onde os agentes não propõe seus planos individuais mas sim partes de um conjunto de planos possíveis que são posteriormente refinados a fim de se ajustarem ao objetivo global. Estes refinamentos são propostos após um processo de busca heurística (usando A*) pelo melhor conjunto de ações conjuntas.

Métodos baseados em planejamento hierárquico também foram propostos: Corkill propôs uma versão distribuída do NOAH e Durfee propôs uma variante na qual cada agente representa seu plano local em vários níveis de abstração. Em cada nível é representado se os conflitos com outros planos / agentes estão resolvidos ou se é preciso passar a um nível mais baixo de detalhe.

3.15.4. Intenções conjuntas

As técnicas até então vistas têm um desempenho razoável em cenários estáticos e/ou onde os agentes têm pouca ou nenhuma incerteza sobre o ambiente (outros agentes incluídos). Infelizmente este não é o caso da maior parte dos cenários de interesse, que são dinâmicos e portanto podem obstruir um trabalho conjunto coerente uma vez que os agentes podem ter visões inconsistentes do ambiente. Em tais cenários, métodos baseados em simples ajuste de planos pré-existentes ou mesmo criados *on-the-fly* para uma situação particular, não mais funcionam pois a dinâmica do ambiente impede a reusabilidade.

[Tambe 1997] propôs um modelo de cooperação entre um time (teamwork) denominado STEAM, que se baseia na hipótese de que para melhorar a reusabilidade de planos e ações existentes, é preciso dotar os agentes de um modelo geral de *teamwork*. STEAM é baseado na teoria de intenções conjuntas (j*oint intentions*) de [Cohen and Levesque 1991]. Este trabalho explicita a diferença entre o agente ter uma intenção individual associada a um objetivo particular e o agente ser parte de um time, o qual tem uma espécie de intenção coletiva associada ao mesmo objetivo.

Na teoria de intenções conjuntas, um time Θ intenciona uma ação conjunta se os membros de Θ comprometem-se a completar uma ação do time, além de também acreditarem que tal ação está sendo executada. Um comprometimento conjunto é definido como um objetivo conjunto persistente (JPG, de *joint persistent goal*). $JPG(\Theta, p, q)$ significa que o time Θ objetiva p (uma ação) e q é uma cláusula que, quando não valer e/ou for não acreditada para algum agente, permite que o time abandone o JPG.

É o caso de $JPG(\Theta, p, q)$ se e somente se três condições forem satisfeitas:

1. todos os membros de Θ acreditam mutuamente que p é falsa;
2. todos os membros de Θ têm p como objetivo mútuo ou seja todos sabem que todos desejam que p seja verdadeira;
3. todos os membros de Θ acreditam que, até que p seja dada como mutuamente atingida, não atingível ou irrelevante, é o caso que p é um objetivo fraco de cada membro μ de Θ; ter p como objetivo fraco implica que se μ (individualmente) descobre que p foi atingido, não é atingível ou é irrelevante, μ se compromete a tornar esta crença individual uma crença mútua.

Um exemplo ilustrativo é o de dois agentes A e B com objetivo de levantar um (mesmo) objeto pesado. Neste caso ambos têm a intenção individual de levantar o objeto mas teamwork precisa ser mais que apenas isto. Suponhamos que A venha a acreditar que o objeto não pode ser levantado. Neste caso, A pode simplesmente abandonar a intenção. Entretanto, teamwork supõe que A passe a ter como objetivo (dai o nome de objetivo fraco) informar B que o objetivo não pode ser alcançado.

STEAM estende estas noções através da construção de uma hierarquia de intenções conjuntas, intenções individuais e crenças sobre outras intenções, bem como preve reorganização do time em caso de falha na execução ou mudanças no ambiente.

Além dos trabalhos de [Cohen and Levesque 1991] e [Tambe 1997], diversos outros formularam o problema dentro de uma abordagem baseada em BDI. Estas abordagens facilitam o projeto dos times mas, novamente, não tratam de forma eficiente a questão da dinamicidade do ambiente. Para tanto, a melhor abordagem é através de MDP's ou, melhor dito, das versões multiagente e com observação parcial de MDP's que serão discutidas na próxima seção. Antes de

passar a esta discussão, fica o registro de um trabalho particularmente interessante pois combina BDI e MDP: [Nair and Tambe 2005].

3.16. Aprendizado e tomada de decisão em ambientes multiagentes

3.16.1. A motivação da teoria de jogos

Muitos sistemas multiagentes podem ser formalizados em termos da teoria de jogos na medida em que as interações entre os agentes podem ser abstraídas através do uso de matrizes de ganho associadas a um conjunto de ações *conjuntas* possíveis. A teoria de jogos pode ser vista como uma extensão da teoria da decisão para o caso particular onde há mais de uma pessoa envolvida no processo de tomada de decisão. Trata-se de uma teoria normativa cujo objetivo é recomendar a jogadores ações que os levarão a maximizar as suas utilidades em cada situação. Esta recomendação é feita com base nos chamados conceitos de solução de um jogo. Um destes conceitos é o de solução-equilíbrio (por exemplo equilíbrio de Nash). Soluções-equilíbrio têm a propriedade que nenhum jogador tem ganhos maiores se desviar dela, ou seja se optar por outra solução.

Em suma, uma vez que os conflitos de interesse podem surgir em interações entre agentes, um processo de coordenação entre eles tem um papel fundamental no processo de decisão. A teoria de jogos oferece uma ferramenta formal para os agentes encontrarem a melhor forma de jogar um jogo. Isto se dá através de análise de algumas hipóteses como um determinado nível de conhecimento comum (com seus problemas conforme visto na Seção 3.13.3) e a racionalidade dos agentes ou seja todos buscam maximizar seus ganhos. Apoiando-se nestas duas hipóteses, é possível que cada jogador se coordene com os demais sem necessidade de comunicação e negociação.

Há diversos tipos de jogos; devido à limitação de espaço, estes serão apenas mencionados brevemente. Um jogo de soma zero é aquele no qual a soma das recompensa de todos os jogadores é zero, ou seja, no caso de 2 jogadores, o ganho de um é exatamente igual à perda do outro. Jogos de soma não zero não apresentam esta propriedade, podendo a soma das recompensa ser qualquer valor real. Um caso particular de jogos de soma não zero é o de jogos com recompensa idêntica (*common reward game* ou *team game*). Nestes, todos os jogadores têm a mesma recompensa, dispensando portanto a representação do ganho individual. Este tipo de jogo é interessante em projeto de sistemas multiagentes para modelar situações onde times de agentes são recompensados por uma ação coletiva.

Como será visto nas próximas subseções, a comunidade de sistemas multiagentes vem “emprestando” da teoria de jogos uma série de metáforas e formalismos, estes já estabelecidos há décadas e portanto, sólidos. A razão é clara: as interações em teoria de jogos são inerentemente multiagente (não há jogo de apenas um agente!) e existe todo um ferramental matemático para analisar desde tomada de decisão em encontros únicos (*one-shot game*), até

a dinâmica do aprendizado em jogos repetidos (*repeated games*) e jogos estocásticos (*stochastic games*). Neste capítulo, apenas estes últimos serão abordados por terem uma relação clara com aprendizado multiagente.

Jogos repetidos e jogos estocásticos na verdade referem-se a um único formalismo com uma variante, como veremos a seguir. Embora o termo *stochastic games* seja popular tanto em teoria de jogos como em sistemas multiagentes, neste texto será usado o termo equivalente MMDP (multiagent MDP) pois este não apenas associa claramente MMDP como uma extensão para MDP, como também inclui o termo multiagente, ideia que o termo jogo estocástico não passa.

3.16.2. MMDP

Como visto na Seção 3.5.2, existem algumas propostas de extensão para o caso de MDP simples. Uma delas (POMDP) foi discutida naquela seção. Aqui será introduzida a extensão para um MMDP.

A generalização de um MDP para n agentes é representada pela tupla MMDP$= (\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ ou seja além dos conjuntos de estados $\mathcal{S}$ e de ações $\mathcal{A}$, da função de recompensa $\mathcal{R}$ e do conjunto das probabilidades de transição $\mathcal{P}$, há o conjunto de agentes $\mathcal{N} = 1..., i, ..., n$. Além disto também cabe observar que, dependendo da abordagem, os conjuntos $\mathcal{S}$ e $\mathcal{A}$ precisam ser estendidos. $\mathcal{A}$ torna-se $\mathcal{A} = \times_{i \in \mathcal{N}} \mathcal{A}^i$ ou seja incluí o conjunto de ações *conjuntas* possíveis dentro do espaço de ações. O mesmo vale para o conjunto $\mathcal{S}$ embora em jogos estocásticos frequentemente se assuma que todos os agentes estão no mesmo estado (mesma matriz de payoff).

Em geral, dependendo das ações realizadas pelos agentes (ou de alguma mudança no ambiente), ocorre uma transição que determina um novo estado ou seja uma nova matriz de recompensa.

Em teoria de jogos é também bastante utilizada a noção de jogo de matriz única que, para efeito de MMDP significa um único estado ou seja a cardinalidade de $\mathcal{S}$ é 1. Neste caso usa-se o termo jogo repetido. Como o formalismo permanece o mesmo, esta distinção é as vezes opaca.

Por fim, como dito na Seção 3.5.2, MDP's não apenas lidam com um único agente, como também assumem que este tem observação total do ambiente. A questão de existência de vários agentes foi discutida acima. Resta a questão da observação parcial. Um dos formalismos que permite modelar este tipo de problema é a de jogos Bayesianos (*Bayesian games*), também chamado de POMDP multiagente. Nestes, os agentes não necessariamente observam seus estados e/ou os ganhos que recebem por suas ações. Este problema apresenta uma grande complexidade pois os POMDP's em si ainda carecem de abordagens eficientes. Sua combinação com a estocasticidade inerente de ambientes com mais de um agente é um desafio em aberto.

Os quatro formalismos para tomada de decisão vistos neste capítulo (MDP e POMDP para cenários monoagente e MMDP e jogos Bayesianos para os casos multiagente) estão fortemente associados a aprendizado (seja mono, seja

multiagente) conforme foi visto na Seção 3.6. A seguir será explorada a conexão entre MMDP e aprendizado por reforço multiagente (doravante abreviado MARL)[11].

3.16.3. Aprendizado por reforço em ambientes multiagente

Quando um agente é colocado em um cenário multiagente, algumas das hipóteses fundamentais do aprendizado de máquina são violadas. O agente não está mais aprendendo a extrapolar a partir de exemplos vindos de um conjunto fixo, mas sim ele próprio é o conceito-alvo que, portanto, muda no tempo. A situação só não é pior porque o conceito-alvo não muda de forma totalmente aleatória mas sim em função da dinâmica do aprendizado de outros agentes, a qual, felizmente, pode ser aprendida.

Entretanto tal aprendizado está longe de ser trivial. Já para o caso monoagente, textos clássicos como [Sutton and Barto 1998] mencionam o problema da maldição da dimensionalidade (*curse of dimensionality*), ou seja o crescimento exponencial do número de parâmetros a serem aprendidos, que cresce em função do número de estados e do número de ações. Em MARL devemos adicionar também o número de agentes.

Diversas soluções para este problema já foram propostas para o caso monoagente (aproximação de funções, abstração em diversos *flavors*, aprendizado hierárquico, etc.). Até aqui, no caso multiagente, a "solução" mais popular tem sido a de fazer com que os agentes aprendam de forma individual ou independente (basicamente utilizando RL monoagente). Esta abordagem tem dois tipos de efeitos: ou o aprendizado se torna bastante ineficiente (por exemplo a convergência ocorre mas em um tempo proibitivamente elevado), ou o aprendizado é completamente ineficaz, como é o caso em alguns problemas de coordenação. Duas instâncias deste tipo de problema aparecem em teoria de jogos e robótica.

Em robótica, um problema recorrente é o de dois robôs tendo que se coordenar para utilizar um recurso de forma não simultânea (por exemplo uma passagem que só pode ser utilizada por um robô de cada vez).

Em teoria de jogos, uma metáfora popular é a de jogo de coordenação (batalha dos sexos, jogos de coordenação pura, etc.). Não é preciso entrar em detalhes sobre a formulação destes jogos. Para efeitos deste texto basta observar que eles têm mais de um equilíbrio. Com isto, agentes que aprendem independentemente não têm bom desempenho pois frequentemente não fazem escolhas coordenadas. No caso de mais de um equilíbrio com a mesma recompensa o agente não consegue distinguir quais ações *conjuntas* devem ser realizadas. Uma discussão sobre esta classe de problemas aparece em

[11] É interessante observar que, na literatura, MDP's aparecem tanto associados aos formalismos em torno de planejamento – para o caso monoagente estocástico como justificado na Seção 3.5.2 – como aos formalismos para RL; já no caso multiagente a primeira conexão praticamente inexiste sendo que os MMDP's são praticamente apenas citados em conexão com MARL.

[Claus and Boutilier 1998] que exploraram apenas os casos destes jogos em suas formas repetidas (um estado). Posteriormente o caso de várias matrizes ou estados foi discutido em diversos artigos. Uma visão geral sobre estas abordagens pode ser encontrada em [Panait and Luke 2005].

Este problema é ainda mais grave quando se trata de ambientes com centenas ou milhares de agentes tentando aprender a dinâmica do jogo, como é o caso em jogos do tipo *congestion games*, *minority games* e *dispersion games*. Para este problema em particular uma boa visão geral aparece em [Tumer and Wolpert 2004].

Mais recentemente [Fulda and Ventura 2007] isolaram 3 fatores que degradam o desempenho de MARL: convergência individual sub-ótima, *action shadowing*, e o problema da seleção de equilíbrio.

Em aprendizado (em geral) um dos principais objetivos é a exploração de um ambiente no qual cada agente tem apenas uma visão parcial ou visão local. Nestes cenários pode-se projetar agentes que aprendem sobre seus respectivos ambientes, ainda que a partir de uma visão parcial, e que posteriormente trocam informações com outros agentes de forma a aumentar as visões locais a fim, por exemplo, de decidir cooperativamente quais partes do ambiente necessitam maior exploração. Em cenários de aprendizado competitivo por outro lado, os agentes tentam maximizar suas próprias utilidades as custas dos demais, não sendo esperado nenhuma forma de cooperação a não ser aquele que traga benefício para o(s) agente(s).

A seguir, a discussão será focada em formalismos para MARL que são baseados em MMDP's e, consequentemente em teoria de jogos, uma vez que esta vertente é considerada menos ad-hoc por alguns pesquisadores. Na Seção 3.16.3.2 será feita uma discussão breve sobre outros métodos.

Aprendizado em sistemas de dois ou mais jogadores tem uma longa tradição em teoria de jogos. A conexão entre sistemas multiagentes e teoria de jogos no que se refere a aprendizado tem sido portanto muito explorada já que as interações em teoria de jogos são inerentemente multiagente, ainda que na maioria das vezes somente dois agentes sejam considerados. Logo, parece natural que a comunidade em torno de MARL explore os formalismos já definidos em teoria de jogos, como por exemplo aqueles baseados em MMDP's. Apesar dos resultados obtidos, há uma discussão em progresso na área sobre se este é o formalismo adequado para MARL como pode ser visto em [Shoham et al. 2007, Stone 2007].

Os problemas de MARL são inerentemente mais complexos que aqueles onde apenas um agente atua. Isto ocorre porque, enquanto um agente está tentando modelar seu ambiente (incluindo outros agentes), os demais agentes podem estar fazendo o mesmo, introduzindo modificações neste ambiente, o que o torna inerentemente não-estacionário. Com isto, pelo menos no caso geral, as garantias de convergência dos métodos de RL (como por exemplo para Q-learning, visto na Seção 3.6.1.2) deixam de valer[12].

Este aumento de complexidade tem diversas consequências. Nos deteremos aqui naquelas que são relacionadas com o formalismo de MMDP. Em primeiro lugar, as abordagens propostas para o caso de jogos de soma geral requerem que várias hipóteses sejam feitas em relação à estrutura da matriz de recompensas, como por exemplo o conhecimento que os agentes têm dela, etc. Devido a estas hipóteses, os resultados relacionados à convergência se restringem a jogos de soma zero ou a jogos de recompensa idêntica, que são aqueles nos quais todos os agentes têm o mesmo ganho. Muitos destes jogos têm mais de um equilíbrio e portanto não é claro qual destes os agentes devem aprender a selecionar.

Em segundo lugar, apesar do formalismo elegante para MMDP's, estes não podem ser empregados no caso de um grande número agentes e/ou de ações, especialmente no caso dos agentes modelarem ações conjuntas. Este problema ocorre porque a complexidade é exponencial no espaço de ações conjuntas. Desta forma, quanto maior o número de ações, maior o espaço de combinações destas. Devido a este fato, a literatura reporta tentativas de desenvolver algoritmos para MARL que são eficientes apenas para 2–3 agentes, 2–3 ações e, não raro, apenas um estado.

Em terceiro lugar, o projetista de aprendizado baseado no formalismo de MMDP precisa decidir se a função $\mathcal{R}$ recompensará os agentes em relação ao ótimo do sistema ou em relação ao ótimo local. Que tipos de problemas admitem um e outro caso? Em alguns domínios é natural que cada agente perceba sua própria recompensa (por exemplo, no tráfego veicular ou mesmo em redes peer2peer), enquanto que em outros como futebol de robôs há uma clara tendência a recompensar cada agente pelo desempenho do time como um todo. Em suma, esta é uma questão conhecida e em aberto. Uma excelente introdução a este tema pode ser encontrada em [Tumer and Wolpert 2004] .

Até aqui estas três questões têm impedido a difusão do uso de MMDP's em problemas do mundo real. Alguns artigos que exploram desafios e caminhos para soluções podem ser encontrados em [Stone and Veloso 2000] (domínio do futebol de robôs, entre outros), [Bazzan 2009] (controle de tráfego), bem como em [Panait and Luke 2005] (vários domínios). Desta forma, há uma busca por soluções, possivelmente heurísticas, que possam ser combinadas com o formalismo básico introduzido na Seção 3.16.2.

3.16.3.1. Abordagens baseadas em MMDP

Nesta seção serão mencionados apenas trabalhos seminais e/ou que trazem um resultado expressivo em termos teóricos como por exemplo convergência para equilíbrio, dado o volume de publicações na área de MARL.

Conforme já colocado, [Claus and Boutilier 1998] tratam da dinâmica do processo de aprendizado em jogos de coordenação através de Q-learning

[12] Aqui vale observar que a despeito deste fato, Q-learning continua sendo um método popular em MARL sendo que sua aplicação é similar à vista naquela seção.

adaptado para o caso de um único estado. Eles comparam o aprendizado individual e o que eles denominaram de JAL (*joint action learners*) onde os agentes aprendem usando modelos que incluem explicitamente as ações conjuntas.

Deve-se a Littman dois resultados importantes. Em [Littman 2001] ele apresenta o algoritmo Friend-or-Foe Q-learning (FFQ) que é capaz de aprender um equilíbrio de Nash dadas algumas condições do jogo. Entretanto este algoritmo necessita saber de antemão se os agentes são amigos ou inimigos ou seja se trata-se de aprendizado entre agentes cooperativos ou competitivos.

O caso particular de jogos de soma zero foi formalizado por [Littman 1994a] (minimax Q). Já o caso geral de soma não zero ainda permanece em aberto. Uma tentativa importante é o trabalho de [Hu and Wellman 1998] (Nash-Q). Houve avanços significativos posteriores, ainda que a maioria dos trabalhos assuma que os agentes conhecem as ações escolhidas pelos demais. Para uma visão mais atualizada, o leitor pode consultar as referências citadas em [Panait and Luke 2005, Shoham et al. 2007].

3.16.3.2. Abordagens não baseadas em MMDP

Na seção anterior, foram mencionados os principais trabalhos que se apóiam em teoria de jogos e MMDP. Entretanto estes formalismos estão longe de ser consenso na área de MARL, e muito menos na área de aprendizado multiagente em geral, o qual transcende aprendizado por reforço.

Algumas críticas feitas são: i) a extrema preocupação com a convergência para pontos de equilíbrio (enquanto se sabe que nem todos os equilíbrios trazem um ganho social mais elevado como no caso do dilema do prisioneiro); ii) o uso de metáforas da teoria de jogos (dilema do prisioneiro, batalha dos sexos, jogos de coordenação, etc.) que pouco têm a ver com o mundo real; iii) hipótese de informação completa e observação total.

O grupo em torno de M. Veloso tem explorado de forma eficaz e eficiente abordagens não baseadas em MMDP, notadamente para o domínio do futebol de robôs. Devido à questão da explosão combinatorial do espaço de ações e estados conjuntos, a qual é agravada em ambientes com um alto número de agentes, a maioria das abordagens segue na linha de abstração deste espaço a fim de reduzir a necessidade exploração do agente.

Uma outra linha de trabalho é a de [Guestrin et al. 2002] em torno de grafos de coordenação, que explora dependências entre os agentes. A partir desta ideia diversas outras exploram a esparsidade das interações entre os agentes de várias formas.

3.17. Para saber mais sobre os tópicos da parte III

Para se aprofundar em alguns dos tópicos apresentados na Parte III, o leitor pode recorrer à diversas fontes bibliográficas. Devido ao fato de que a relação (e comentário) destas tornaria este capítulo excessivamente longo, optou-se por disponibilizar este texto em um arquivo separado disponível no site `http:`

//www.inf.ufrgs.br/~bazzan/downloads/anexo_psmp3.pdf.

3.18. Conclusão

Existem diversos livros texto sobre IA no mercado, inclusive nacional, que tratam de forma competente esta subárea da ciência da computação. Isto também é parcialmente verdade em relação à área de sistemas multiagentes embora aqui haja uma carência de textos em português. Entretanto, a proposta do presente capítulo é mostrar a IA sob um ângulo diferente ou seja um ângulo multiagente, uma vez que a autora acredita que a IA clássica, monoagente não apenas difere significativamente da IA multiagente como também tem dificuldades em atender os requisitos colocados pela Internet e pelos sistemas distribuídos e colaborativos, todos de natureza essencialmente social.

Desta forma, a Parte III deste capítulo procurou descrever uma versão multiagente de quatro dentre as áreas básicas da IA: representação de conhecimento, resolução de problemas (aqui incluindo planejamento), tomada de decisão e aprendizado. Por versão multiagente, entenda-se uma IA onde não apenas um agente age, mas sim onde vários agentes interagem bem como têm conhecimento parcial sobre o problema.

Antes, na Parte I foi feita uma introdução a estas mesmas áreas, dentro da visão clássica, monoagente. Na Parte II foi feita uma breve introdução aos conceitos sobre agentes autônomos e sistemas multiagentes necessários para a compreensão da Parte III.

Agradecimentos

Este trabalho somente foi possível graças ao apoio das agências financiadoras dos projetos de pesquisa ligados a este tema, tanto os ainda em andamento quanto os já encerrados. Desta forma agradeço ao CNPq pelo apoio aos projetos de pesquisa, bem como ao programa de bolsas de pesquisa e bolsas de pós graduação de diversos orientandos. Agradeço ainda à CAPES e à Fundação Alexander von Humboldt pelo apoio ao projeto de cooperação internacional com a Alemanha e à bolsa de pós doutoramento, respectivamente. Não menos importante, agradeço aos meus ex-alunos que contribuíram com suas pesquisas em algumas das áreas abordadas neste texto, notadamente Bruno Castro da Silva, Paulo Roberto Ferreira Jr., Fernando dos Santos e Daniela Scherer dos Santos, cujas teses e dissertações são próximas ao presente material. Por fim agradeço aos revisores de partes deste texto, anônimos e voluntários (como Filipo S. Peroto e Rafael H. Bordini).

References

[Bazzan 2009] Bazzan, A. L. C. (2009). Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. *Autonomous Agents and Multiagent Systems*, 18(3):342–375.

[Bittencourt 2001] Bittencourt, G. (2001). *Inteligência Artificial: Ferramentas e Teorias*. Editora da UFSC, Florianópolis, 2a. edition.

[Bond and Gasser 1988] Bond, A. H. and Gasser, L. (1988). “Readings in distributed artificial intelligence”. In *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, San Mateo, California.

[Brooks 1986] Brooks, R. (1986). A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23.

[Claus and Boutilier 1998] Claus, C. and Boutilier, C. (1998). “The dynamics of reinforcement learning in cooperative multiagent systems”. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 746–752.

[Cohen and Levesque 1990] Cohen, P. R. and Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–261.

[Cohen and Levesque 1991] Cohen, P. R. and Levesque, H. J. (1991). Teamwork. *Noûs*, 25(4):487–512.

[Durfee 1999] Durfee, E. H. (1999). “Distributed problem solving and planning”. In Weiß, G., editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 121–164. MIT Press, Cambridge, MA, USA.

[Fikes and Nilsson 1971] Fikes, R. and Nilsson, N. J. (1971). “STRIPS: A new approach to the application of theorem proving to problem solving”. In *Proc. of the IJCAI*, pages 608–620.

[Fulda and Ventura 2007] Fulda, N. and Ventura, D. (2007). “Predicting and preventing coordination problems in cooperative Q-learning systems”. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 780–785.

[Gasser and Huhns 1990] Gasser, L. and Huhns, M. N., editors (1990). *Distributed artificial intelligence: vol. 2*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Georgeff 1983] Georgeff, M. P. (1983). “Communication and interaction in multi-agent planning”. In *Proc. of the Nat. Conf. on Art. Intelligence*, pages 125–129. AAAI Press.

[Guestrin et al. 2002] Guestrin, C., Lagoudakis, M. G., and Parr, R. (2002). “Coordinated reinforcement learning”. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML)*, pages 227–234, San Francisco, CA, USA. Morgan Kaufmann.

[Hu and Wellman 1998] Hu, J. and Wellman, M. P. (1998). “Multiagent reinforcement learning: Theoretical framework and an algorithm”. In *Proc. 15th International Conf. on Machine Learning*, pages 242–250. Morgan Kaufmann.

[Jennings 1996] Jennings, N. R. (1996). “Coordination techniques for distributed artificial intelligence”. In O’Hare, G. M. P. and Jennings, N. R., editors, *Foundations of Distributed Artificial Intelligence*, pages 187–210. John Wiley & Sons, New York.

[Littman 1994a] Littman, M. L. (1994a). "Markov games as a framework for multi-agent reinforcement learning". In *Proceedings of the 11th International Conference on Machine Learning, ML*, pages 157–163, New Brunswick, NJ. Morgan Kaufmann.

[Littman 1994b] Littman, M. L. (1994b). "Memoryless policies: theoretical limitations and practical results". In *Proceedings of the third international conference on Simulation of adaptive behavior: from animals to animats 3*, pages 238–245, Cambridge, MA, USA. MIT Press.

[Littman 2001] Littman, M. L. (2001). "Friend-or-Foe Q-learning in general-sum games". In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML01)*, pages 322–328, San Francisco, CA, USA. Morgan Kaufmann.

[Malone and Crowston 1994] Malone, T. and Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119.

[Modi et al. 2003] Modi, P. J., Shen, W.-M., Tambe, M., and Yokoo, M. (2003). "An asynchronous complete method for distributed constraint optimization". In *Proc. of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 161–168, New York, USA. ACM Press.

[Nair and Tambe 2005] Nair, R. and Tambe, M. (2005). Hybrid BDI-POMDP framework for multiagent teaming. *Journal of Artificial Intelligence Research*, 23:367–420.

[Panait and Luke 2005] Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

[Petcu and Faltings 2005] Petcu, A. and Faltings, B. (2005). "A scalable method for multiagent constraint optimization". In Kaelbling, L. P. and Saffiotti, A., editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 266–271, Edinburgh, Scotland. Professional Book Center.

[Rosenschein and Zlotkin 1994] Rosenschein, J. and Zlotkin, G. (1994). *Rules of Encounter*. The MIT Press, Cambridge (MA).

[Russel and Norvig 1995] Russel, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice-Hall.

[Russell and Norvig 2004] Russell, S. and Norvig, P. (2004). *Inteligência Artificial*. Campus, Rio de Janeiro, RJ. Tradução da segunda edição.

[Shoham and Leyton-Brown 2009] Shoham, Y. and Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

[Shoham et al. 2007] Shoham, Y., Powers, R., and Grenager, T. (2007). If multi-agent learning is the answer, what is the question? *Artificial Intelligence*,

171(7):365–377.

[Silva et al. 2006] Silva, B. C. d., Basso, E. W., Bazzan, A. L. C., and Engel, P. M. (2006). “Dealing with non-stationary environments using context detection”. In Cohen, W. W. and Moore, A., editors, *Proceedings of the 23rd International Conference on Machine Learning ICML*, pages 217–224. New York, ACM Press.

[Stone 2007] Stone, P. (2007). Multiagent learning is not the answer. It is the question. *Artificial Intelligence*, 171(7):402–405.

[Stone and Veloso 2000] Stone, P. and Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383.

[Sutton and Barto 1998] Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

[Tambe 1997] Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR)*, 7:83–124.

[Tumer and Wolpert 2004] Tumer, K. and Wolpert, D. (2004). “A survey of collectives”. In Tumer, K. and Wolpert, D., editors, *Collectives and the Design of Complex Systems*, pages 1–42. Springer.

[Wainer 1994] Wainer, J. (1994). “Yet another semantics of goals and goal priorities”. In *Proc. of the ECAI*, pages 269–273.

[Watkins 1989] Watkins, C. (1989). *Learning from Delayed Rewards*. PhD thesis, University of Cambridge.

[Weiss 1999] Weiss, G. (1999). *Multiagent Systems - A modern Approach to Distributed Artificial Intelligence*. The MIT Press, Cambridge, MA.

[Wooldridge 2002] Wooldridge, M. J. (2002). *An Introduction to Multiagent Systems*. John Wiley & Sons, Chichester.

[Wooldridge 2009] Wooldridge, M. J. (2009). *An Introduction to MultiAgent Systems*. John Wiley & Sons, Chichester. Second edition.

[Yokoo et al. 1992] Yokoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K. (1992). “Distributed constraint satisfaction for formalizing distributed problem solving”. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, pages 614–621.