Trabalho de Formatura Supervisionado
Planejamento Probabilístico

Helton Massato Kishi
Supervisora: Leliane Nunes de Barros

2 de dezembro de 2007

1 Introdução

Em Inteligência Artificial, Processos de Decisão Markovianos (*Markov Decision Processes* - MDPs) são utilizados para resolver problemas de planejamento sob incerteza. A tarefa de planejamento pode ser definida como: dado um conjunto de ações, um estado inicial e um conjunto de estados meta, encontrar uma seqüência de ações que leve um agente do estado inicial a um dos estados meta.

Planejamento sob incerteza é uma extensão do problema de planejamento clássico, em que as ações podem ter efeitos incertos. Nesse caso, o estado atual em que o agente se encontra e a escolha da ação a ser tomada naquele momento, determinam a distribuição de probabilidade do próximo estado a ser visitado e o objetivo do planejador é criar uma *política*, que mapeia estados em ações e que maximize as chances de se chegar aos estados finais a partir de um estado inicial.

1.1 Motivação

Existem várias aplicações para o planejamento probabilístico modelados como MDPs, entre elas:

- escalonamento de elevadores;
- planejamento de rotas de viagem;
- automação de processos de manufatura;
- automatização de navegação de aviões.

1.2 Objetivos do trabalho

Este trabalho tem como objetivo introduzir os conceitos e soluções existentes para o problema de MDP. Serão explicados e implementados (em Java) três algoritmos que resolvem o problema de MDP: iteração por valor, *Real-Time Dynamic Programming* e *Labeled Real-Time Dynamic Programming*. Além disso, será feita uma análise comparativa entre os algoritmos acima, usando-se um problema exemplo que será explicado ao longo desta monografia.

2 Planejamento em Inteligência Artificial

Planejamento é o processo de escolher e organizar ações através da antecipação de seus efeitos, tendo como objetivo satisfazer uma meta pré-estabelecida [AIMA]. O processo de planejamento tem como entrada um conjunto de estados, um conjunto de ações e uma função de transição de estados, que mapeiam para cada estado e ação, qual será o próximo estado. A solução de um problema de planejamento é um *plano*, que é um conjunto de ações necessárias para desempenharmos alguma tarefa ou atingir algum objetivo.

2.1 Planejamento Determinístico

Planejamento determinístico, também chamado de planejamento clássico, é o caso mais simples de planejamento. Chama-se determinístico pois não há nenhuma incerteza envolvida sobre os efeitos das ações, ou seja, o efeito de uma ação será sempre o mesmo. Mais precisamente, sempre que a ação a_i for executada no estado s_i, chegaremos ao estado s_j. Dado o estado inicial, o conjunto de estados meta, o planejador deve devolver uma seqüência de ações que leva o sistema do estado inicial até o estado final.

2.2 Planejamento sob incerteza

Planejamento sob incerteza é quando o efeito de uma ação pode não ser sempre o mesmo. Mais precisamente, quando a ação a_i for executada, existirá um conjunto de próximos estados possíveis. Este tipo de planejamento pode ser subdividido em dois tipos de planejamento: o *planejamento probabilístico* e o *planejamento não-determinístico*.

No planejamento probabilísitico, os efeitos de uma ação são representados por uma matriz de probabilidade. Para cada ação, existe uma distribuição de probabilidade dos efeitos da ação. Esta matriz de probabilidade, dentre as várias possibilidades de próximo estado, define de forma quantitativa quais são os estados com maior possibilidade de serem o próximo estado. Este é o problema que será tratado neste trabalho.

Existe também o planejamento não probabilísitico, também chamado de planejamento não determinístico. Neste tipo de planejamento, ao contrário do probabilístico, não existe uma quantificação das probabilidades. Só sabemos que a execução de uma ação a em um estado s nos leva a um conjunto de próximos estados possíveis. Esse tipo de incerteza, quando não se conhece as probabilidades, é também chamado de *incerteza knightiana*.

Há também uma variação do planejamento probabilístico, em que os dois tipos de incerteza co-existam: oplanejamento com imprecisão na definição das probabilidades.

A solução para o problema de planejamento sob incerteza é uma política que mapeia estados em ações. Assim, o agente deve, após executar uma ação, observar em que estado ele se encontra para poder então poder identificar a próxima ação indicada pela política.

3 Planejamento probabilístico utilizando MDP

MDP [Boutilier] [Putterman], do inglês *Markov Decision Process*, é um modelo matemático que pode ser utilizado para resolver o problema de planejamento probabilístico. Um MDP pode ser definido pela tupla $< S, A, P, R, C >$, sendo:

- S, um conjunto de estados;
- A, um conjunto de ações;
- $P : A \times S \times S \rightarrow [0, 1]$ uma função de transição de estados, onde $P(a, s, s')$ é a probabilidade do agente ir do estado $s \in S$ para o estado $s' \in S$ após a execução da ação $a \in S$;
- $R : S \rightarrow \mathbf{R}$ uma função de recompensa. $R(s)$ é a recompensa obtida se o sistema chegar ao estado s.
- $C : S \times A \rightarrow \mathbf{R}$, uma função do custo. $C(s, a)$ é o custo de se executar a ação $a \in A$ no estado $s \in S$.

3.1 Estados e transição de estados

O estado nada mais é do que uma descrição do sistema num dado momento [Boutilier]. Um estado pode ser representado, por exemplo, através de um conjunto de variáveis valoradas de estado (modelo *fatorado*). Pode-se também definir MDPs simplesmente enumerando-se todos os estados possíveis do sistema, sem explicitarmos os valores das variáveis de estado (modelo *enumerativo*). Todos os algoritmos estudados neste trabalho, usam o modelo *enumerativo*.

A soma das probabilidades de transição de um dado estado, executando-se uma dada ação, deve ser 1. Mais precisamente, seja S um espaço de estados e A um espaço de ações,

$$\forall s \in S, \forall a \in A, \sum_{s' \in S} P(a, s, s') = 1$$

Note que a transição de estados do planejamento clássico também pode ser representada usando-se esta função. No entanto, para cada estado e ação, apenas *um* dos estados tem probabilidade 1 e os outros 0.

Podemos representar (graficamente) um MDP através de grafos dirigidos, um para cada ação. Cada grafo representa a transição de estados para uma dada ação. Cada nó do grafo é um estado e de cada estado saem arestas para outros estados. Cada aresta tem um valor variando de 0 a 1, representando a probabilidade de sairmos do estado origem da aresta e chegarmos no estado destino da aresta. A soma de todas as arestas saindo de um mesmo nó deve ser sempre 1.

3.2 Trajetória e história do sistema

Os termos *trajetória* e *história* são utilizados para descrever o comportamento do sistema durante o momento de solucionar o problema [Boutilier]. A história completa de um sistema é a seqüência de estados, ações e observações de estados gerada desde o estado inicial até um certo instante de tempo t de interesse, podendo ser tanto finita, quanto infinita. Uma história pode ser representada por:

$$(< S_0, O_0, A_0 >, < S_1, O_1, A_1 >, \ldots, < S_t, O_t, A_t >)$$

onde O_i é dada pela funcão de observabilidade $O : S \rightarrow O_i$.

No caso de sistemas totalmente observáveis, O é a função identidade e para descrever o histórico é necessário apenas os estados em que o sistema passou e as ações tomadas, ou seja:

$$(< S_0, A_0 >, < S_1, A_1 >, \ldots, < S_{t-1}, A_{t-1} >, S_t)$$

3.2.1 Função de custo e recompensa

A função de custo, $C : S \times A \rightarrow \mathbf{R}$, indica o custo de aplicarmos a ação a no estado s. Nesse trabalho, custo é sempre maior ou igual a zero. [Boutilier]

A função recompensa $R : S \rightarrow \mathbf{R}$ indica a recompensa de estar no estado s. Nesse trabalho, as recompensas assumirão valores positivos.

Estas duas funções são utilizadas para definirmos a função valor, que será definida na seção a seguir.

3.3 Função valor

Para que o agente consiga ver a qualidade das ações tomadas, é necessária uma função valor $V : H_s \rightarrow \mathbf{R}$ [Boutilier] [Putterman]. O agente prefere a história h ao invés de h' se $V(h) < V(h')$.

Para o cálculo da função de valor, é necessário explicitar um tempo T de duração da simulação, chamado de *horizonte*. Dado um horizonte T (que pode corresponder ao número de ações de uma história), podemos calcular a função valor utilizando as funções de custo e recompensa definidos na seção anterior, da seguinte forma:

$$V(h) = \{\sum_{t=0}^{T-1} C(s_t, a_t) - R(s_t)\} - R(s_T).$$

Podemos também calcular a função valor com um horizonte infinito [Boutilier]:

$$V(h) = \sum_{t=0}^{\infty} (\gamma^t (C(s_t, a_t)) - R(s_t))$$

onde γ é a taxa de desconto ($0 \leq \gamma < 1$). Quanto menor o γ, menor será a importância dos acontecimentos longe do início.

Se fizermos a soma de todos os custos e recompensas para todos os estados da história h_s, estaremos calculando uma função de valor *time-separable* [Boutilier]. No entanto, pode existir a necessidade dos custos e recompensas serem variáveis ao longo do tempo de execução do sistema. Isto pode ser contornado criando-se mais estados, um para cada período de tempo.

3.4 Política

Para se encontrar uma solução para o problema de MDP é necessário achar uma *política* $\pi : S \rightarrow A$ [Boutilier], que define para cada estado qual deve ser a melhor ação a ser tomada. Os algoritmos apresentados a seguir recebem como entrada um MDP e devolvem uma política calculada a partir da definição da função valor.

4 Algoritmo de iteração por valor: um método para resolver um MDP

Resolver um MDP é o problema de encontrar uma política $\pi : S \rightarrow A$. Uma política ótima π^* indica qual é a melhor ação que o agente deve executar em cada estado do MDP.

O algoritmo IV garante encontrar uma política total ótima usando programação dinâmica. Para encontrarmos tal política é preciso escolher ações

que minimizem o valor da função de valor com o horizonte infinito. Isto é calculado a partir da *equação de Bellman* [Bellman], dada por:

$$V(s) = \min_{a \in A}\{C(a,s) - R(s) + \gamma \sum_{s' \in S} P(a,s,s')V(s')\}$$

A política ótima pode ser extraída pela equação:

$$\pi^*(s) = \arg\min_{a \in A}\{C(a,s) - R(s) + \gamma \sum_{s' \in S} P(a,s,s')V(s')\}$$

Note que $\pi^*(s)$ é igual a ação que minimiza a equação de bellman.

A idéia do algoritmo é inicializar um vetor $V(s)$ com valores arbitrários. A cada iteração, usa-se a equação de Bellman para atualização dos valores de $V(s)$ para todo $s \in S$ e calcula-se o valor do residual, que é a diferença entre os valores de $V(s)$ antes e depois da iteração.

Se o valor do residual for menor do que ϵ, um valor muito pequeno conhecido a priori, o algoritmo termina, obtendo assim a política ótima. Caso contrário, o algoritmo executa novamente a iteração.

Segue abaixo o pseudo-código do algoritmo de iteração por valor.

IteracaoPorValor($S, A, P, R, C, \gamma, \epsilon$)

```
1  Inicializa um vetor V(s) com valores arbitrários.
2  n ← 0
3  while ∃s ∈ S tal que Residual(s) > ε
4        do
5           ▷ Calcula valores de V(s) para todo estado s.
6           for all s ∈ S
7                 do
8                    temp ← min_{a∈A}{C(a,s) − R(s) + γ Σ_{s'∈S} P(a,s,s')V(s')}
9                    Residual(s) = |V(s) − temp|
10                   V(s) ← temp
11                   π(s) ← arg min_{a∈A}{C(a,s) − R(s) + γ Σ_{s'∈S} P(a,s,s')V(s')}
12          n ← n + 1
13 return π
```

5 SSP: Caminho Estocástico Mínimo

Quando se deseja resolver problemas de planejamento em Inteligência Artificial, deve se considerar algumas informações adicionais:

- s_0, estado inicial.
- $S_G \subset S$, conjunto dos estados meta.

Assim, um problema de planejamento probabilístico pode ser naturalmente descrito como um problema de SSP [**?**] [Putterman]. O SSP (do inglês *Shortest Stochastic Path*) é um sub-problema do problema de MDP em que, além das definições do MDP, é necessário definir o estado inicial s_0 e o conjunto dos estados meta S_G e para o qual se procura uma política parcial ótima.

6 Algoritmos para resolvermos um SSP

6.1 O algoritmo RTDP

O algoritmo RTDP (Real-Time Dynamic Programming) resolve o problema de SSP, devolvendo uma política parcial [RTDP]. O algoritmo inicializa os valores de V(s) como no algoritmo de iteração por valor. A cada iteração, o algoritmo recalcula o valor de V(s) apenas para o estado atual usando a equação de Bellman. Após esta etapa, o algoritmo faz com que o agente execute (através de uma simulação) a melhor ação dada pela política parcial construída até o momento e atualiza o estado atual. O algoritmo termina quando alcançamos algum dos estados meta em S_G. Como o critério de parada do algoritmo é a chegada do agente até algum dos estados meta, este algoritmo não garante devolver uma política ótima. Além disso, o fato do algoritmo atualizar o valor de V(s) apenas para os estados s visitados pela simulação, o número de atualizações de V(s) é menor do que o calculado pelo algoritmo de iteração por valor. Assim, há estados para os quais nunca se calcula o V(s).

Segue abaixo o pseudo-código do RTDP:

RTDP(S, A, P, R, C, s_0, S_G)
1 Inicializa um vetor $V(s)$ com valores arbitrários.
2 $s \leftarrow s_0$
3 **while** $s \notin S_G$
4 **do**
5 $V(s) \leftarrow \min_{a \in A}\{C(a,s) - R(s) + \sum_{s' \in S} P(a,s,s')V(s')\}$
6 $\pi(s) \leftarrow \arg\min_{a \in A}\{C(a,s) - R(s) + \sum_{s' \in S} P(a,s,s')V(s')\}$
7 $s \leftarrow$ executeAction($\pi(s)$)

Onde executeAction($\pi(s)$) é uma função que faz a simulação da execução da ação $\pi(s)$ no estado atual do sistema e devolve qual é o próximo estado. Esta simulação pode ser feita criando-se partições no intervalo $[0, 1]$ a partir das probabilidades definidas pela função de transição. Ao se gerar um número aleatório, basta descobrir a qual intervalo este número pertence para sabermos qual será o próximo estado.

No algoritmo RTDP, podemos observar uma melhoria de desempenho pois só atualizamos os estados relevantes, ou seja, aqueles que são mais

visitados. Uma das desvantagens do RTDP é que ele não calcula uma política ótima pois o algoritmo pára quando chegamos em um estado final.

6.2 O algoritmo LRTDP

Um dos problemas do algoritmo RTDP é que os estados só são atualizados de acordo com a simulação, ou seja, quanto maior a probabilidade de irmos para um estado s, mais vezes iremos recalcular o valor de $V(s)$. Além disso, o algoritmo RTDP não devolve uma política ótima.

O algoritmo LRTDP [Geffner], *Labeled Real-Time Dynamic Programming*, introduz uma heurística ao algoritmo RTDP e devolve uma política parcial ótima, ou seja, devolve uma política ótima somente para os estados relevantes para chegarmos desde o estado inicial até algum dos estados meta.

Esta heurística é chamada de *Labeling procedure* [Geffner], e é feita usando-se o método CHECKSOLVED() [Geffner], cujo pseudo-código é dado a seguir.

checkSolved(s, ϵ)

```
 1  rv ← TRUE
 2  open ← CRIAPILHAVAZIA()
 3  closed ← CRIAPILHAVAZIA()
 4
 5  if s.SOLVED = FALSE
 6     then open.EMPILHA(s)
 7
 8  while open ≠ PilhaVazia
 9        do s ← OPEN.DESEMPILHA()
10           closed.EMPILHA(s)
11           ▷ Checa o Residual...
12           if Residual(s) > ε
13              then rv ← FALSE
14                   continue;
15
16           ▷ expande o estado..
17           a ← π(s)
18           for each s' tal que P(a, s, s') > 0
19                do if s'.SOLVED = FALSE and s' ∉ open ∪ closed
20                      do open.empilha(s')
21
22  if rv = TRUE
23     then for each s' ∈ closed
24                do s'.SOLVED ← TRUE
25     else
26                do while closed ≠ PilhaVazia
27                      do s ← CLOSED.DESEMPILHA()
28                         S.UPDATE()
29  return rv
```

O método CHECKSOLVED(S, ϵ) faz uma busca em largura a partir do estado s procurando por algum estado que tenha o residual maior do que ϵ. Quando ele acha um estado com tal condição, o CHECKSOLVED(S, ϵ) devolve falso e recalcula o valor de $V(s)$ para todos os estados que foram visitados durante a busca em largura. Dado um estado s, os seus filhos são todos os estados s' tais que $P_a(s'|s) > 0$, onde a é a ação dada pela política no momento da busca.

O método CHECKSOLVED(S, ϵ) devolve verdadeiro se e somente se para todos os estados s' alcançáveis por s, $Residual(s') < \epsilon$. É possível provar que usando o algoritmo abaixo, resolvemos o problema de MDP [Geffner].

```
1  while s0.SOLVED = FALSE
2        do CHECKSOLVED(s0, ε)
```

No entanto, neste algoritmo, não há a tentativa de fazer checkSolved(s', ϵ) para $s' \neq s_0$. Além disso, sabemos que o estado s_0 costuma ser o último estado a convergir. Os estados mais próximos aos estados meta são aqueles que têm maior probabilidade de convergir mais rapidamente.

Portanto, o algoritmo LRTDP funciona da seguinte maneira, um pouco diferente do algoritmo acima:

```
LRTDP(ϵ)
1  ▷ Calcula V(s) uma vez para todos os estados.
2  for each s ∈ S
3        do Update(s);
4
5  s ← s_0
6  while s.SOLVED = FALSE
7        do LRTDPTrial(s, ϵ)
```

```
LRTDPTrial(s, ϵ)
 1  visited ← criaPilhaVazia
 2  while s.SOLVED = FALSE
 3        do visited.empilha(s)
 4
 5           if s ∈ S_G
 6              then break;
 7
 8           a ← π(s)
 9           s.update();
10           s ← executeAction(π(s))
11
12  while visited ≠ PilhaVazia
13        do s ← visited. desempilha();
14           if checkSolved(s, ϵ)
15              then break;
```

O algoritmo basicamente calcula em um primeiro momento o valor de V(s) uma única vez para todos os estados. Após esta etapa, ele chama a função LRTDPTrial até que o estado inicial seja resolvido.

No LRTDPTrial, inicialmente usa-se o mesmo método do RTDP até que um estado que já tenha convergido (residual menor do que ϵ) seja encontrado ou quando a simulação atinge um estado final. Todos os estados visitados durante este processo são colocados em uma pilha, na medida em que vão sendo visitados.

Quando um estado que converge é encontrado ou quando a simulação chega em algum estado meta, percorremos esta lista na ordem inversa em que seus elementos foram adicionados. Para cada elemento desta lista, é

feito um CHECKSOLVED(S, ϵ). Se o checkSolved devolver verdadeiro, ele pega o próximo elemento da lista. Caso contrário, o método LRTDPTRIAL termina.

O algoritmo só termina quando o CHECKSOLVED(s_0, ϵ) devolver verdadeiro, ou seja, quando todos os estados atingíveis do estado s_0 tenham convergido. Assim, podemos garantir que o algoritmo calcula uma política ótima para os estados atingíveis pelo estado inicial s_0.

Este algoritmo faz a seguinte suposição: Todos os estados meta são atingíveis a partir de todos os estados, caso contrário, ele falha.

Os resultados da implementação do LRTDP feita nesse trabalho podem ser vistas na seção 8.

7 Exemplo: o robô entregador de cartas

Esta seção tem como objetivo mostrar um exemplo de um problema de planejamento probabilístico que será utilizado para comparação dos algoritmos implementados.

Considere um robô que faz a entrega de cartas em um andar de escritórios dividido em três salas: a sala do chefe, do funcionário e da secretária. O robô deve decidir, em um dado instante, para quem deve entregar a carta.

O robô nem sempre consegue obter sucesso na entrega das cartas, pois existe a possibilidade do destinatário não estar presente na sala quando o robô tenta fazer uma entrega. Sejam p, q e r, respectivamente, as probabilidades de encontrarmos a secretária, o funcionário ou o chefe em sua sala em um dado momento.

Há também prioridades diferentes. É muito mais importante entregar a carta ao chefe do que entregar a carta para a secretária.

O estado pode ser definido usando três variáveis booleanas, uma para cada destinatário, indicando se o robô entregou ou não a carta para o respectivo destinatário. São eles HD0 (Have Delivered 0), HD1 (Have Delivered 1) e HD2 (Have Delivered 2), sendo o destinatário 0 a secretária, 1 o funcionário e 2 o chefe.

O custo da ação é definido pelo custo de energia gasta pelo robô para entregar a carta (que é a mesma para todos os destinatários). A recompensa é definida de acordo com a prioridade de entrega.

As ações que o robô pode executar são:

- **d0** entregar carta para a secretária.
- **d1** - entregar carta para o funcionário.
- **d2** - entregar carta para o chefe.

Probabilidades:

- p = prob. de encontrarmos a secretária em sua sala = 0,9

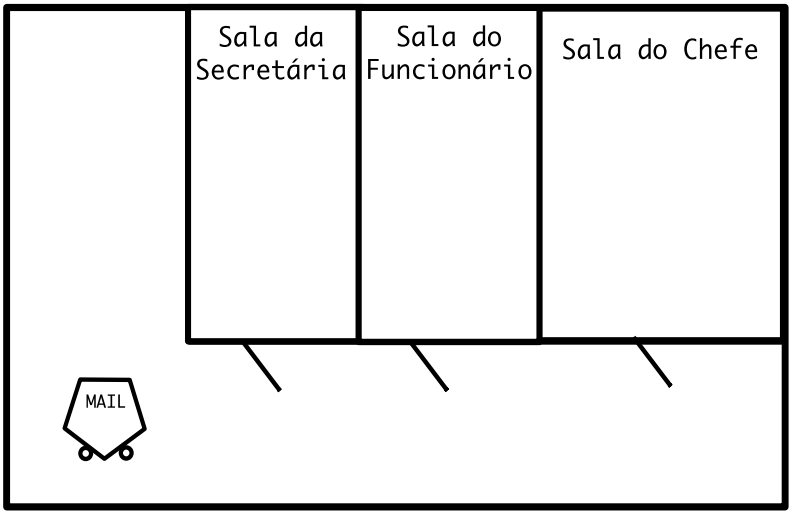

Figura 1: O robô deve decidir para quem ele deve entregar a carta.

- q = prob. de encontrarmos o funcionário em sua sala = 0,7
- r = prob. de encontrarmos o chefe em sua sala = 0,5

A figura 2 ilustra, através de um grafo, a matriz de transição de estados. Uma aresta com rótulo *d0/p*, indo de um estado s para s', significa que a probabilidade de irmos do estado s para o estado s' aplicando a ação *d0* é p.

A recompensa é calculada da seguinte maneira:

$$R(s) = HD0 * 1 + HD1 * 2 + HD2 * 4$$

onde HD0 = 1 se a variável booleana HD0 for verdade, e HD0 = 0 se HD0 for falso.

Por exemplo, o estado $s1$, em que apenas a variável HD0 é verdade, tem a recompensa igual a 1. Assim, a matriz de recompensa é definida conforme a tabela 1.

Estado	s0	s1	s2	s3	s4	s5	s6	s7
Recompensa	0	1	2	4	3	5	6	7

Tabela 1: Recompensa do problema do Robô

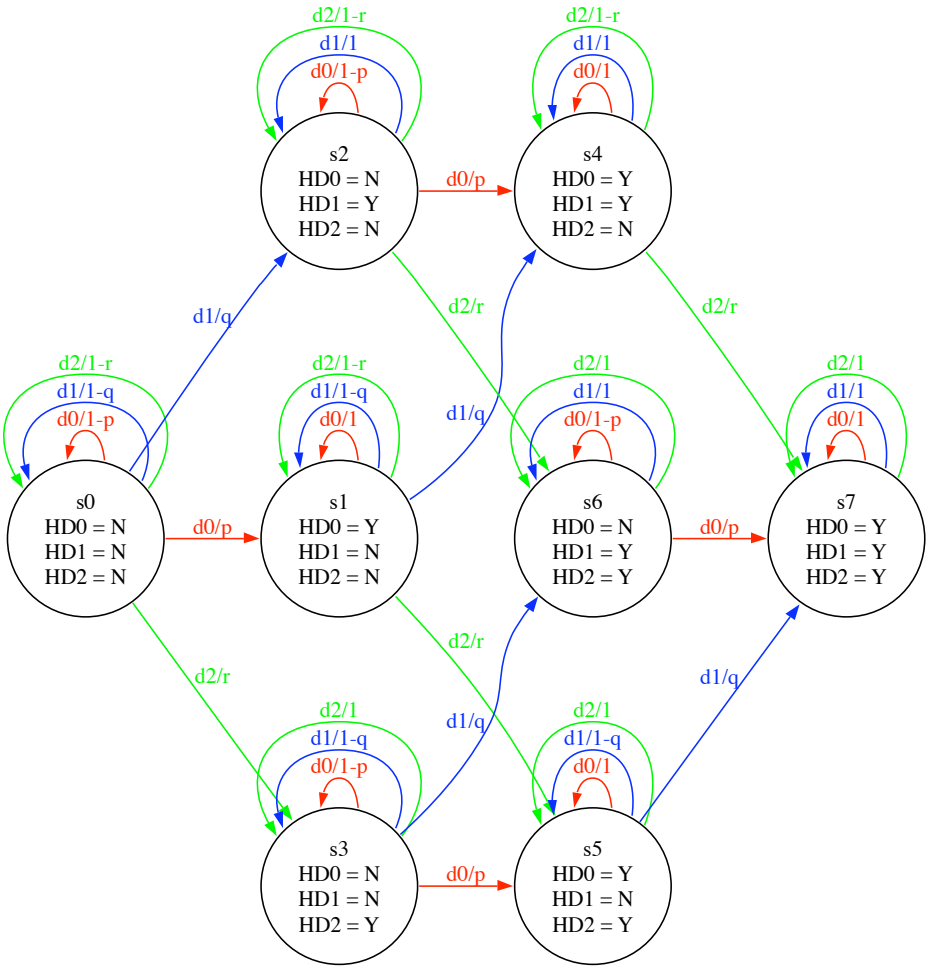

Figura 2: Grafo de transição do problema do robô que representa a matriz de probabilidade

O custo de todas as ações é igual a 10.

Podemos descrever facilmente a melhor política para este problema. Se o estado inicial for aquele em que nenhuma carta ainda foi entregue, o robô deve entregar a carta para o chefe, uma vez que essa é a ação com maior prioridade. Quando a carta do chefe já foi entregue, o robô deve entregar a carta para o funcionário. O robô deverá entregar a carta para a secretária apenas quando as cartas do chefe e do funcionário forem entregues. Além disso, o robô deve levar em consideração as probabilidades dos destinatários estarem em suas respectivas salas. Dependendo dessas probabilidades o robô pode decidir violar as prioridades de entrega para conseguir realizar seus objetivos de forma mais eficiente. Por exemplo, o robô entregaria a carta primeiro para a secretária caso a probabilidade dela estar na sala fosse 1 e a do chefe for 0.1. Para as probabilidades definidas nesta seção, para cada estado definido, a melhor ação é entregar a carta para o destinatário com maior prioridade que ainda não recebeu a carta. Essas probabilidades foram escolhidas de forma a permitir uma análise comparativa entre os algoritmos implementados que salientassem as principais características de cada estratégia.

8 Análise comparativa

Para analisar os algoritmos estudados neste trabalho, resolvemos o MDP modelado para o problema do robô entregador de cartas usando os três algoritmos: IV, RTDP e LRTDP. Na tabela 2 mostramos as políticas obtidas.

Política IV	
Estado	Ação
s0	d2
s1	d2
s2	d2
s3	d1
s4	d2
s5	d1
s6	d0
s7	d0

Política RTDP	
Estado	Ação
s0	d0
s1	d1
s2	null
s3	null
s4	d2
s5	null
s6	null
s7	null

Política LRTDP	
Estado	Ação
s0	d2
s1	d0
s2	d0
s3	d1
s4	d2
s5	d1
s6	d0
s7	d0

Tabela 2: Políticas geradas pelos algoritmos implementados para o problema do robô entregador de cartas.

A primeira análise será sobre o número de vezes que cada estado foi atualizado. A segunda análise será sobre a relação entre o número de atualizações com a qualidade das soluções.

8.1 Número de atualizações

Nesta seção será analisado o número de atualizações de $V(s)$ (isto é, o número de vezes em que $V(s)$ é calculado para o estado s) realizadas pelos algoritmos IV, RTDP e LRTDP. A tabela 3 mostra os resultados obtidos:

Estado	IV	RTDP	LRTDP
s0	12	1	62
s1	12	2	1
s2	12	0	1
s3	12	0	16
s4	12	2	1
s5	12	0	1
s6	12	0	7
s7	12	0	1
Totais	96	5	90

Tabela 3: Comparação do número de atualizações de $V(s)$ entre os algoritmos IV, RTDP e LRTDP.

É possível observar que o algoritmo que faz menos atualizações é o RTDP. Isto ocorre pois o algoritmo RTDP atualiza apenas os estados alcançados pela simulação. Além disso, o algoritmo termina quando o sistema chega a um dos estados meta, ao contrário dos dois outros algoritmos que checam o valor do residual.

Já o algoritmo de IV faz o maior número de atualizações. Isto ocorre devido ao fato do IV calcular, a cada iteração, os valores de $V(s)$ para todos os estados, terminando apenas quando o valor do residual for menor do que ϵ para todo $s \in S$. Por isso, o número de atualizações é o mesmo em todos os estados.

O número de atualizações feitas pelo algoritmo LRTDP está entre os de IV e RTDP. No entanto, podemos observar que há alguns estados com apenas uma atualização, feita na hora de inicializarmos o algoritmo do LRTDP.

8.2 Número de atualizações *versus* Qualidade

Pudemos observar que o algoritmo RTDP fez o menor número de atualizações nos valores de $V(s)$. No entanto, a qualidade da política gerada é muito pior do que as políticas geradas pelo IV e LRTDP. O algoritmo RTDP não garante que os valores de $V(s)$ tenham convergido, gerando apenas uma política parcial que não é ótima.

O algoritmo IV calcula uma política total ótima, sendo ele o algoritmo que devolve políticas com a melhor qualidade. No entanto, tem o pior desempenho, pois é o algoritmo que faz o maior número de atualizações.

No caso do LRTDP, o algoritmo calcula a política ótima apenas para os estados que fazem parte de um caminho possível do estado inicial para algum estado meta. Isto pode ser observado comparando a política gerada pelo LRTDP e o IV. Note que a política gerada pelo LRTDP é igual à gerada pelo IV (tabela 2) nos estados em que o número de atualizações é maior do que 1 (tabela 3). Os estados que só foram atualizados uma única vez, são os estados que foram atualizados apenas na inicialização do LRTDP, em que é necessário calcular em uma única iteração o valor de V(s) para todo estado $s \in S$. Portanto, os estados com apenas uma única atualização foram desconsiderados pelo algoritmo, ou seja, não fazem parte de um caminho possível do estado inicial para o estado meta. Assim, o algoritmo LRTDP calcula a política ótima para os estados com número de atualizações maior do que 1. No problema de planejamento para o robô entregador de cartas, definimos probabilidades e recompensas de forma que a política ótima estabelece uma ordem entre as ações, fazendo com que alguns estados não sejam alcançados pela política ótima parcial. Por exemplo, os estados $\{s1, s2, s4, s5, s7\}$, não fazem parte da política ótima parcial caclulada pelo LRTDP. De fato, ao executarmos a política ótima total calculada pelo IV, a partir do estado s0, os estados $\{s1, s2, s4, s5, s7\}$ nunca serão alcançados.

9 Conclusão

Soluções clássicas de planejamento probabilístico usando MDPs geram políticas totais, isto é, mapeiam ações para todo estado do MDP e possuem complexidade $O(|S|^2|A|)$ [Papadimitriou].

Para problemas de planejamento, em que são dados o estado inicial e o estado meta, o problema do MDP se reduz a um problema do tipo SSP, isto é, o problema de se encontrar o caminho estocástico mínimo. Dado um SSP, estamos interessados em achar uma política que leve o agente do estado inicial até algum estado meta. Assim, é necessário apenas calcular a política ótima para os estados relevantes ao problema dado de planejamento, ou seja, estados que fazem parte de um caminho estocástico possível, do estado inicial até algum estado meta.

Em geral, essa é uma característica fundamental dos domínios de problemas de planejamento com espaço de estados e ações finita mas muito grande. Na grande maioria dos problemas a solução desejada é uma política parcial.

O interesse desse trabalho foi o de mostrar como podemos encontrar políticas para problemas de planejamento baseados em MDPs evitando cálculos desnecessários, isto é, devolvendo uma política parcial ótima. Para isso, foi feita uma análise comparativa entre três algoritmos: o IV, RTDP e LRTDP. O primeiro resolve um problema de MDP, enquanto os outros dois um pro-

blema de SSP.

O algoritmo RTDP calcula de uma maneira rápida uma política parcial. No entanto, esta solução não é ótima, pois este algoritmo não garante a convergência dos valores de $V(s)$.

Com a implementação do algoritmo LRTDP, pudemos observar para o exemplo do robô, que o algoritmo devolve a política ótima apenas para os estados relevantes, obtendo assim uma política parcial ótima.

10 Parte subjetiva sobre a realização do trabalho

10.1 Desafios e frustrações

Um dos maiores desafios encontrados foi a falta de tempo disponível para elaborar este trabalho. Neste último ano, tive que conciliar os estudos com o estágio. Houve momentos em que tive que dedicar muito tempo para o estágio, e conseqüentemente o tempo dedicado ao trabalho foi comprometida.

Apesar da teoria de MDPs ser conhecida desde a década de 60 e ter sido usada pela engenharia para modelar processos estocásticos, tive dificuldade também em encontrar uma aplicação real para o problema de planejamento baseado em MDPs.

10.2 Disciplinas cursadas no BCC mais relevantes para o trabalho

Segue abaixo a lista de disciplinas que me ajudaram muito para fazer esta monografia:

- **MAC0122 - Princípios de Desenvolvimento de Algoritmos**
- **MAC0338 - Análise de Algoritmos**
  - Estas duas disciplinas foram a base para que eu pudesse entender, analisar e implementar algoritmos.
- **MAC0425 - Inteligência Artificial**
  - Esta foi a disciplina com a qual tive o primeiro contato a respeito de Inteligência Artificial e o problema de planejamento em IA. Foi também quando conheci a Leliane, a orientadora desta monografia.
- **MAE0228 - Noções de Probabilidade e Processos Estocásticos**
  - Nesta disciplina, me familiarizei com cadeias de Markov, o que ajudou muito para o entendimento de MDPs.

10.3 Disciplinas cursadas no BCC que considero importantes para minha formação

Segue abaixo a lista de disciplinas que não me ajudaram diretamente para a elaboração desta monografia mas que contribuíram muito para a minha formação.

- **MAC0424 - O computador na sociedade e empresa**

  - Esta é uma disciplina muito diferente das outras matérias do IME, que são puramente técnicas. O objetivo da disciplina é mostrar as conseqüências sociais do uso de computadores.

- **MAT0213 - Álgebra II**
  - Considero esta disciplina muito importante para a minha formação pois faz com que os alunos tenham uma maior capacidade de abstração matemática, com a generalização de inúmeros conceitos matemáticos.

- **EAD0630 - Matemática aplicada a Finanças**
  - Esta disciplina foi muito interessante pois pude observar como a matemática é aplicada na área de finanças. Embora a parte matemática seja bem simples, aprendi muitos conceitos relacionados a finanças e economia.

Referências

[Boutilier] C. Boutilier, T. Dean, S. Hanks. *Decision-Theoretic Planning: Structural Assumptions and Computational Leverage.* Journal of Artificial Intelligence Research 11, 1-94, 1999.

[Geffner] H. Geffner, B. Bonet. *Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming.* Proceedings of Thirteenth International Conference on Automated Planning and Scheduling, Trento, Italy. AAAI Press, 12-21, 2003.

[RTDP] Andrew G. Barto, Steven J. Bradtke, Satinder P. Singh. *Learning to act using real-time dynamic programming.* Technical Report UM-CS-1993-002, 1993.

[AIMA] Stuart Russell, Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall; 2nd Edition.

[Automated Planning] Malik Ghallab, Dana Nau, Paolo Traverso. *Automated Planning: theory and practice* Morgan Kaufmann Publishers, May 2004, 663 pages.

[Putterman] M. L. Putterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, New York, 1994.

[Papadimitriou] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Bellman] R. E. Bellman. *Dynamic Programming.* Princeton University Press, 1957.