2º Projeto IA 2018/2019 – Relatório

Grupo tp017 – João Neves 83405, Diogo Redin 84711

Parte 1 - Redes Bayesianas

A primeira parte do projeto é constituída por dois problemas principais:

1. Obter a probabilidade conjunta numa rede de Bayes, segundo uma evidência, com a função **computeJointProb**.
2. Obter a probabilidade a posteriori de um evento segundo uma evidência, ou seja, realizar uma inferência exata na rede de Bayes, com a função **computePostProb**.

Para este efeito, são utilizadas as seguintes classes:

- **Classe Node**: representa um nó da Rede Bayesiana. Tem complexidade espacial $O(2^{k+1})$, sendo *k* o número de nós pais diretos. A complexidade temporal da criação é $O(n.log(n))$, sendo o tempo necessário para formatar a lista obtida como argumento para uma lista simples unidimensional.
- **Classe BN**: representa a Rede Bayesiana. Tem complexidade espacial $O(n.2^n)$, sendo este o tamanho ocupado pela lista de **Nodes**.
- **Classe Factor**: representa um fator utilizado para o algoritmo de eliminação de variáveis. Tem complexidade espacial $O(2^n)$, sendo este o número de variáveis das quais depende.

Probabilidade Conjunta

A resolução da probabilidade conjunta é trivial, sendo apenas necessário obter a coordenada específica da lista de probabilidades segundo a evidência. Na implementação específica deste projeto, a operação tem complexidade temporal $O(k.n)$, em que *k* é o número de nós na rede, e *n* é o número de pais de cada nó.

Probabilidade A Posteriori

A resolução da probabilidade a posteriori requer a realização de uma inferência exata na rede de Bayes. Para este problema, é utilizado o **Algoritmo de Eliminação de Variáveis**, cuja complexidade temporal é de $O(2^n)$, em que *n* é o número de nós presentes na rede.

A realização do algoritmo tem 3 partes específicas:

1. **Eliminação de variáveis de evidência**, com complexidade temporal $O(k.2^n)$. *n* é o número de pais de cada nó que irá sofrer esta operação, e *k* será o número de nós.
2. **Eliminação de variáveis desconhecidas** (*Point-wise Product* & *Sum Out*), com complexidade temporal $O(k.2^n)$, em que *k* é o número de operações de *Point-wise Product* que irão ser realizadas, e 2^n é o número de variáveis dos quais os fatores resultantes irão depender. Visto que os fatores resultantes do *Point-wise Product* podem ter mais variáveis que quaisquer fatores individuais, este é a fase em que a complexidade espacial e temporal é definida, visto que o aumento de espaço e tempo é exponencial para fatores maiores.
3. **Point-wise Product** dos restantes fatores, que terão unicamente dependência da *query variable*, devido às restantes operações. Deste modo, os fatores resultantes terão todos o mesmo tamanho. Após a multiplicação, o fator é normalizado, e é obtido o resultado da *query*.

Vantagens

A utilização do algoritmo de eliminação de variáveis, enquanto que tem teoricamente complexidade temporal e espacial de crescimento exponencial, na prática, permite obter resultados de inferências em tempo possivelmente linear. Dependendo da ordem de eliminação pode haver melhor complexidade computacional para queries.

Desvantagens

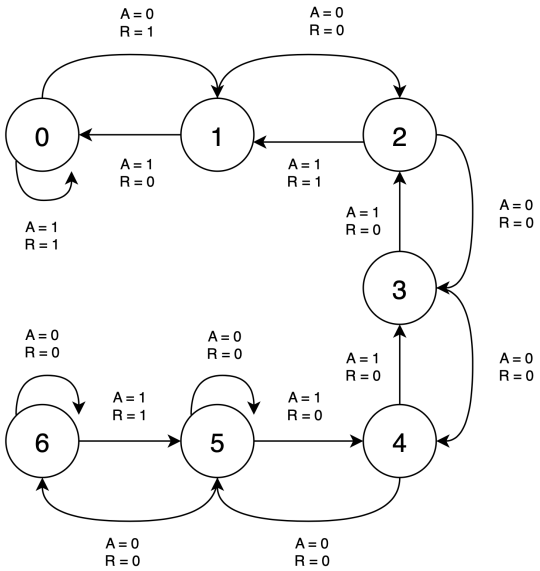
Na implementação específica ao projeto, a ordem de eliminação é semi-arbitrária, sendo

dependente da ordem de inserção de nós no grafo. Enquanto que se o grafo for ordenado de pais para filhos esta ordem é aceitável, poderia haver uma otimização gananciosa para evitar o aumento de tempo e espaço no Point-wise Product, eliminando as variáveis que obtivessem fatores com menor número de variáveis dependentes.

Poderia ser também implementada uma política de ignorar fatores irrelevantes à query específica, ou seja, a eliminação de fatores que não são antecessores de variáveis query ou de variáveis evidência que são irrelevantes para o resultado.

Parte 2 - QLearning

1º Ambiente



Função de Recompensa

R(Estado,Ação)=

		Ações	
		0	*1*
Estados	*0*	1	1
	1	0	0
	2	0	1
	3	0	0
	4	0	0
	5	0	0
	6	0	1

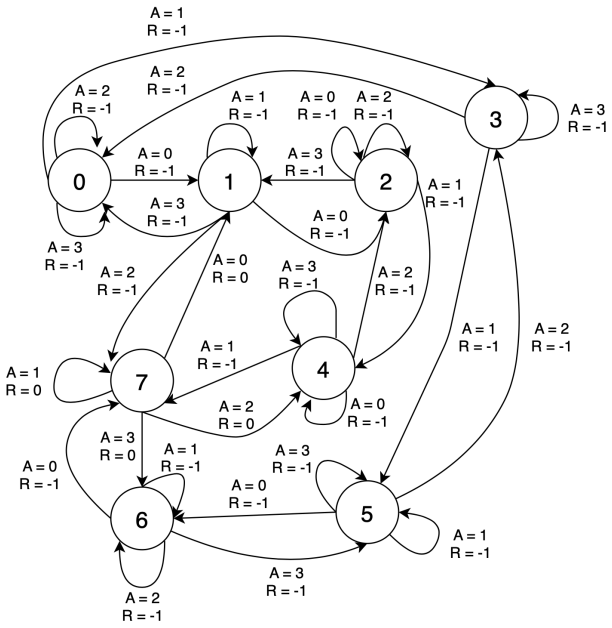
Política Ótima

Avaliando os valores de $V^{\pi}(s_i)$ para *exploration* e *exploitation*, concluímos que a política ótima é *exploitation* pois é a que maximiza as recompensas acumuladas.

Movimento do Agente

Após a aprendizagem é possível observar que o agente começa em 3, desloca-se para 2, depois para 1 e finalmente para 0 onde permanece em ciclo, onde pode obter constantemente uma recompensa de 1.

2º Ambiente



Função de Recompensa

R(Estado,Ação)=

		Ações			
		0	*1*	*2*	*3*
Estados	*0*	-1	-1	-1	-1
	1	-1	-1	-1	-1
	2	-1	-1	-1	-1
	3	-1	-1	-1	-1
	4	-1	-1	-1	-1
	5	-1	-1	-1	-1
	6	-1	-1	-1	-1
	7	0	0	0	0

2º Projeto IA 2018/2019 – Relatório

Grupo tp017 – João Neves 83405, Diogo Redin 84711

Política Ótima

Por análise dos valores de Q da trajetória dada é possível concluir que a política adotada foi *Exploration*, uma vez que os mesmos são em geral menores que 0 indicando que foram calculados para todas as ações possíveis nos diferentes estados.

Movimento do Agente

Por analise da matriz de valores Q e da trajetória é possível observar que o agente tende a deslocar-se para 7 onde se mantém em ciclo, pois tem recompensa de 0 constante (a maior possível no ambiente).

Crítica aos resultados obtidos

Dado o primeiro ambiente e a trajetória obtida consideramos que a aprendizagem foi bem-sucedida pois o agente consegue permanecer constantemente numa situação em que recebe recompensa máxima.

No segundo exemplo é já dada uma trajetória gerada após aprendizagem e são obtidos os valores de qualidade dos pares *(estado, ação)*. É possível observar que a matriz contém os valores de qualidade maiores para as ações do estado 7, que são precisamente as ações que contribuem para uma maior acumulação de recompensa.

Consideramos, portanto, que a função de geração de trajetórias e de avaliação dos valores de Q funcionam corretamente.

Métodos implementados

Trace2Q - Calcula os valores de *Qualidade* das ações tomadas numa dada trajetória. Permite construir uma matriz Q com conhecimento sobre as melhores ações a tomar. Foi implementada a seguinte fórmula de para atualizar os valores de Q:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma\, max_a Q(s',a') - Q(s,a)]$$

Exploitation - Política de escolha de uma ação, que consiste em escolher a ação que para um dado estado tem um *maior valor de Q*. O objetivo foi no final gerar uma trajetória ideal com base nos valores de Q gerados com *exploration* anteriormente.

Exploration - Política de escolha de uma ação, em que é escolhida uma ação aleatória com probabilidade de sucesso superior a 0, com vista à exploração de todas as ações possíveis.

Vantagens, Desvantagens e Limitações da Implementação

A estratégia ε-*greedy* implementada tem como vantagem não necessitar de memorizar dados específicos da exploração do ambiente, ficando tudo implicitamente guardado nos valores de Q. Tem como desvantagem estar dependente de um fator de exploração γ que tem de ser manualmente ajustado para resultados ótimos.

Complexidade computacional e possíveis métodos alternativos

A complexidade do pior caso para atualização de um valor de Q, nesta implementação baseada em tabelas de *(Estados, Ações)*, está diretamente relacionada com a quantidade de estados e ações possíveis. Temos, portanto, complexidade temporal $O(E \times A)$. O espaço também depende diretamente desta quantidade e, portanto, temos complexidade espacial $O(E \times A)$. Existe uma proposta de algoritmo que atrasa o cálculo do valor de Q até este ser necessário, poupando assim cálculos de Q de estados sem interesse. Esses estados sem interesse não se aplicam ao nosso caso em que exploramos a totalidade do ambiente. No entanto, numa implementação em que só explorássemos estados com probabilidade de sucesso relevante, poderia ser interessante poupar esses cálculos. Esta alteração levaria a uma complexidade temporal de pior caso de $O(E)$, no entanto teria como desvantagem podermos chegar a uma solução em que descartaríamos estados com baixa probabilidade de sucesso, mas que levariam a uma melhor recompensa.