Atividade 1: Tutorial 7 – Reinforcement Learning

1) Introdução

Nesta atividade será coberto o tema de *reinforcement learning* (aprendizado reforçado) ou RL sob a perspectiva do SOAR.
Em sumário, um aprendizado reforçado se dá pelo incentivo a um agente a tomar certas ações preferencialmente a outras. Este incentivo é feito por meio de recompensas (*rewards*). Na vida real, é comum utilizar a técnica de aprendizado reforçado com animais, como o cachorro doméstico. Quando o cachorro age de acordo com o que desejado, o treinador dá uma recompensa qualquer, seja alimento, um brinquedo ou um simples afago. Com esta técnica, espera-se que o animal passe a efetuar aquela mesma ação dada uma situação ou estímulos semelhantes àquelas do treinamento.

Na arquitetura SOAR, agimos de forma semelhante. Ao identificarmos que uma ação foi tomada dada uma situação (por exemplo seleção de um operador) podemos prover uma recompensa através da augumentação *reward-link* do estado superior do agente.

O tutorial apresenta o exemplo de um agente muito simples chamado *left-right*. Tudo o que este agente faz é alterar de uma posição inicial *start* para uma posição final (*left* ou *right)* e parar a execução. A cada decisão tomada, uma recompensa positiva ou negativa associada à direção é dada ao agente. Podemos então reiniciar o agente mantendo-se os valores de RL que serão levados em consideração na próxima execução. Abaixo encontra-se um exemplo de como reforçar o aprendizado de um agente através do *reward-link*:

sp {elaborate*reward	
(state <s> ^name left-right	
^reward-link <r>	seja <r> o reward-link
^location <d-name>	Seja <d-name> o nome da direção para onde o agente se moveu
^direction <dir>)	Seja <dir> uma das possíveis direções definidas no estado
(<dir> ^name <d-name> ^reward <d-reward>)	Associa a <d-reward> o valor da recompensa relativa a esta direção, onde a direção é a mesma para onde o agente se moveu.
-->	
(<r> ^reward.value <d-reward>)	Passa o valor da recompensa para o *reward-link*
}	

Podemos ver através do *Soar Debugger*, a quantidade de vezes que cada ação foi tomada e qual o valor atual da tendência que o agente tem de tomar uma das ações. Chamamos cada um destes valores de 'indiferença numérica'.

As tabelas abaixo exibem estes valores para dois agentes diferentes com 10 execuções sequenciais, onde A é a quantidade de vezes que a ação foi tomada e I o valor da indiferença numérica da ação. Internamente, a seleção de uma ou outra ação é feita através da alteração da preferência entre operadores levando-se em consideração os valores de indiferença numérica atuais.

	Quantidade de ações de cada tipo e valor do incentivo por execução																					
Execução	0		1		2		3		4		5		6		7		8		9		10	
	A	I	A	I	A	I	A	I	A	I	A	I	A	I	A	I	A	I	A	I	A	I
Right	0	0	0	0.00	1	0.30	2	0.51	3	0.66	4	0.76	4	0.76	4	0.76	5	0.83	6	0.88	7	0.92
Left	0	0	1	-0.30	1	-0.30	1	-0.30	1	-0.30	1	-0.30	2	-0.51	3	-0.66	3	-0.66	3	-0.66	3	-0.66

	Quantidade de ações de cada tipo e valor do incentivo por execução																					
Execução	0		1		2		3		4		5		6		7		8		9		10	
	A	I	A	I	A	I	A	I	A	I	A	I	A	I	A	I	A	I	A	I	A	I
Right	0	0	1	0.3	2	0.51	3	0.66	4	0.76	5	0.83	6	0.88	7	0.92	8	0.94	9	0.96	10	0.97
Left	0	0	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00

Através das tabelas é possível perceber que existe um grau de incerteza sobre que ação ocorrerá mesmo que haja uma grande diferença entre os modificadores de cada ação. Na primeira tabela, vê-se que a sexta ação foi um movimento para a esquerda, mesmo que o incentivo para a direita seja razoavelmente maior que o outro. Isso acontece por causa do tipo da política de exploração utilizada pelo agente. Neste tutorial, a política de exploração é a *epsilon-greedy,* na qual existe uma percentagem de chance de que outros operadores que não o mais preferido seja escolhido para aplicação. Os parâmetros de cada política podem ser alterados pelo usuário conforme desejado, assim como as políticas em si.

2) Criação de um agente com RL

Existe um padrão o qual deve ser adotado para que uma regra seja reconhecida como atualizável pelo mecanismo de RL. Este padrão é como se segue:

sp {my*proposal*rule	O lado esquerdo da regra pode adotar qualquer forma válida
(state <s> ^operator <op> +	
^condition <c>)	
-->	
(<s> ^operator <op> = 2.3)	O lado direito só pode conter a associação de um único valor de indiferença numérica de valor constant e nada mais
}	

Para que todo par estado-ação tenha um valor de indiferença associado, é necessário escrever uma regra para cada um destes pares. Como exemplo, o tutorial cita o *Water Jug Problem*. Dadas as operações *Fill, Empty* e *Pour*, uma jarra com capacidade 5 e uma jarra com capacidade 3 temos:

- Estados: 4*6 = 24 possíveis combinações de volume existente em cada jarro
- Ações: 3*2 = 6 possíveis ações a cada momento (cada ação tendo cada um dos jarros como alvo)
- Total de pares: 24*6 = 144

Felizmente, contudo, estas regras podem ser geradas de forma parametrizada utilizando o comando gp do SOAR. Abaixo um exemplo desta abordagem no contexto do *Water Jug Problem*:

gp {rl*water-jug*empty	
(state <s> ^name water-jug	
^operator <op> +	
^jug <j1> <j2>)	
(<op> ^name empty	
^empty-jug.volume [3 5])	uma regra para cada jarro
(<j1> ^volume 3	
^contents [0 1 2 3])	uma regra para cada estado do jarro de volume 3
(<j2> ^volume 5	
^contents [0 1 2 3 4 5])	uma regra para cada estado do jarro de volume 5
-->	
(<s> ^operator <op> = 0)	Valor inicial da indiferença numérica deste operador
}	

Apesar de ser necessário criar uma regra nesta forma para cada operador, o mesmo não se faz necessário para o feedback (recompensa). Podemos, por exemplo, fornecer uma recompensa para o agente do *Water Jug Problem* quando o problema for resolvido apenas:

sp {water-jug*detect*goal*achieved	Regra para verificação do objetivo
(state <s> ^name water-jug	O objetivo é alcançado se o jarro de volume 3 tem conteúdo 1
^jug <j>	
^reward-link <rl>)	
(<j> ^volume 3 ^contents 1)	
-->	
(write (crlf) |The problem has been solved.|)	
(<rl> ^reward.value 10)	Recompensa o agente
(halt)}	

3) Conclusão

SOAR permite a utilização de algoritmos de *reinforcement learning* de maneira prontamente aplicável. Se o usuário desejar, existem muitas formas de alterar a forma que o aprendizado se dá, seja pela alteração da política de aprendizado quanto pelos parâmetros de cada política. Para que estas políticas sejam utilizadas da melhor forma é necessário que o usuário entenda bem seu funcionamento e a relação entre seus parâmetros, sem contudo precisar saber a fundo como o algoritmo procede.