MAC5788 - Relatório EP4: Planejamento Probabilístico

Viviane Bonadia NUSP 9167607, Diego Araújo NUSP 7157092,
Ignasi Andrés NUSP 8193481
Universidade de São Paulo
São Paulo, Brazil
{vbonadia, diegoamc, ignasi} @ime.usp.br

22 de Junho de 2015

1 Introdução

O planejamento consiste na elaboração de um plano de ação, ou seja um conjunto de ações, para atingir um determinado objetivo [Russell et al., 1995]. No planejamento probabilístico os efeitos das ações possuem uma incerteza, quando uma ação é executada em um determinado estado, não existe uma certeza sobre qual será o próximo estado. O que sabemos é uma distribuição de probabilidade dos estados alcançáveis. Implementamos e realizamos diversos experimentos com quatro algoritmos para encontrar uma política em problemas de planejamento probabilístico: ILAO*, RTDP, LRTDP, e VI que serão melhor explicados ao longo deste relatório. Este trabalho está organizado da seguinte maneira: na seção 2 descrevemos mais detalhadamente Planejamento Probabilístico, na seção 3 descrevemos cada um dos algoritmos implementados, na seção 4 descrevemos alguns detalhes da nossa implementação, na seção 5 apresentamos os experimentos realizados bem como os resultados obtidos e, por fim, na seção 6 expomos as conclusões obtidas com este trabalho.

2 Planejamento probabilístico

Em problemas de planejamento clássico, executar uma ação em um determinado estado leva o agente para um outro único estado. No planejamento

probabilíıstico, a realização de uma ação pode levar o agente à diferentes estados. Ou seja, não é possível saber com certeza qual será o próximo estado do agente após escolher uma determinada ação. O que conhecemos, em problemas desse tipo, são as probabilidades que o agente tem de, dado um estado s ele alcançar um estado s' ao escolher uma determinada ação.

A solução para problemas de planejamento probabilístico é uma política π. Uma política mapeia os estados do problema em ações de forma a otimizar a escolha das ações segundo algum critério (por exemplo custo ou recompensa).

Podemos resolver problemas de planejamento probabilístico usando Processo de Decisão Markoviano (MDP) para representá-los. Um MDP pode ser definido como uma tupla $\langle S, A, R, P, \gamma \rangle$ onde:

- S é o conjunto finito de estados
- A é o conjunto finito de ações
- R é o modelo recompensa associado a cada par $\langle$estado, ação$\rangle$
- P é o modelo de transição probabilístico. $P(s'|s, a)$ é a probabilidade de um agente que está no estado $s \in S$ ir para o estado $s' \in S$ ao executar a ação $a \in A$
- γ é o fator de desconto $(0 \leq \gamma < 1)$

Um MDP poder ser de horizonte finito ou infinito. Neste segundo caso, o agente não altera a forma de agir com o passar do tempo e seu objetivo é encontrar uma política ótima que maximiza a recompensa descontada num horizonte infinito.

A função valor ótima (V*) é a função que está associada a qualquer política ótima e deve satisfazer a seguinte equação:

$$V^*(a) = \max_{a \in A(s)} \{R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s')\} \quad (1)$$

Existem diversos algoritmos que resolvem este tipo de problema. Os algoritmos implementados neste trabalho são alguns exemplos propostos para problemas de planejamento probabilístico.

3 Algoritmos

3.1 VI

O Algoritmo Iteração de Valor (VI) [Bellman, 1957] garante encontrar uma política ótima usando programação dinâmica.

Na primeira iteração o algoritmo inicia V(s) com valores arbitrários e, a cada iteração, atualiza V(s) para todos os estados segundo a Equação 1. O algoritmo termina quando o valor residual é menor que ε.

O valor residual é a diferença entre os valores $V(s)_i$ (calculados na iteração atual) e $V(s)_{i-1}$ (calculados na última iteração). ε é um valor muito pequeno, definido no algoritmo. Em nossa implementação usamos $\varepsilon = 10^{-7}$.

A cada iteração, este algoritmo atualiza todos os estados e, como o espaço de estados pode ter tamanho exponencial, problemas mais complexos tornam-se inviáveis de resolver.

Em razão da atualização de todos os estados a cada iteração, o VI é chamado de solução síncrona.

3.2 RTDP

O algoritmo RTDP (*Real-Time Dynamic Programming*) [Barto et al., 1995] diferente do algoritmo VI, não garante como solução uma política ótima. Porém, RTDP encontra valor ótimo para todos os estados relevantes. Estados relevantes são aqueles alcançáveis a partir do conjunto de estados inicial segundo uma política ótima.

Assim como o algoritmo VI, ele também inicia $V(s)$ com valores arbitrários porém, a cada iteração o algoritmo recalcula o valor de $V(S)$ apenas para o estado atual usando a Equação 1, por esse motivo RTDP é um exemplo de algoritmo de programação dinâmica assíncrona.

Após a etapa de inicialização, o RTDP executa várias simulações (*trials*). A cada simulação um estado do conjunto de estados inicial é escolhido aleatoriamente. Para obter o próximo estado, o algoritmo sorteia o próximo estado a partir da função de transição $P(.|s, a)$. RTDP termina um trial se encontrar um estado meta ou quando uma profundidade limitada é alcançada.

O RTDP não tem condição de parada. Portanto, em nossos experimentos, consideramos um *timeout* de 1 minuto. Esse valor foi escolhido baseado no tempo de execução do algoritmo LRTP, explicado a seguir.

3.3 LRTDP

O algoritmo LRTDP (*Labeled Real-Time Dynamic Programming*) [Bonet, 2003], baseia-se no algoritmo RTDP. Ele introduz uma heurística ao algoritmo RTDP e, diferente deste, devolve uma política parcialmente ótima. A política é chamada de parcialmente ótima por devolver uma política ótima apenas para os estados relevantes partindo do estado inicial estado inicial até o estado meta.

Enquanto no algoritmo RTPD, quanto maior a probabilidade de um estado s ser alcançado mais vezes seu valor função $V(s)$ será atualizado, no LRTDP os estados que já convergiram são rotulados. A cada novo *trial* o algoritmo irá priorizar a atualização dos estados que ainda não convergiram (e não apenas os caminhos com maior probabilidade) aumentando a velocidade de convergência do algoritmo.

O algoritmo termina quando o estado inicial converge com valor residual menor que um determinado ε. Neste trabalho, consideramos $\varepsilon = 10^{-7}$.

É importante notar que tanto o LRTDP quanto o RTDP tem comportamento *anytime*. Isso significa que eles podem produzir uma boa política rapidamente e melhorá-la com o tempo.

3.4 iLAO*

O algoritmo ILAO* (*Improved LAO**) [Hansen and Zilberstein, 2001], assim como RTDP e LRTP, utiliza programação dinâmica na atualização dos estados.

O princípio deste algoritmo também consiste em criar *trials* ou caminhos entre o estado inicial e um estado meta.

Se chegarmos a um estado meta, o algoritmo executa o algoritmo Value Iteration entre os estados do caminho.

Se o algoritmo encontrar um estado novo que não foi adicionado no caminho, ou algum estado que ainda não convergiu, ele realiza um novo *trial* e assim sucessivamente até que todos os estados no caminho estejam convergidos, e nenhum caminho novo seja encontrado.

O algoritmo *LAO** original executa o algoritmo VI sempre que adiciona um novo estado no caminho. Esta versão do algoritmo é menos eficiente. Por isso, os autores propuseram uma versão na qual a etapa de execução do VI somente é executada quando um caminho é descoberto.

Uma das vantagens deste algoritmo a respeito da família RTDP é que ele analisa não somente os estados com maior probabilidade de serem alcançados, mas todos os estados que podem ser alcançados a partir de uma mesma ação não determinística. Isso pode acelerar o tempo de convergência do algoritmo,

uma vez que, a cada iteração, ele atualiza um maior número de estados.

4 Implementação

Nesta seção, apresentaremos como executar o programa e alguns detalhes de implementação, ressaltando decisões importantes que tomamos durante o desenvolvimento.

4.1 Como executar o programa?

O projeto foi implementado usando a linguagem de programação Ruby. Portanto, para rodá-lo é preciso ter o Ruby (versão acima da 1.9.3) instalado no computador.

Na raiz do projeto (diretório *probabilistic_planning*) execute o seguinte comando:

ruby probabilistic_planning.rb <domínio> <algoritmo>

- <domínio> é o nome do domínio que será usado, os parâmetros aceitos são: **tireworld** ou **navigation_problem**
- <algoritmo> é o nome do algoritmo utilizado para gerar a política, os parâmetros aceitos são: **ILAO**, **LRTDP**, **RTDP** ou **VI**.

Um exemplo do comando é dado a seguir:

ruby probabilistic_planning.rb tireworld ILAO

Esse comando rodará o EP para todos os problemas do domónio Tireworld disponíveis no diretório *problems/tireworld*. Após a execução, os resultados estarão disponíveis no diretório *results/tireworld*. É preciso criar este diretório manualmente para que os resultados sejam corretamente armazenados.

4.2 Detalhes do código

Cada uma das classes e seus principais detalhes de implementação são explicados a seguir.

- **requirements.rb**: classe que inclui todas as dependências do programa.

- **parser.rb**: classe que lê a descrição dos problemas e a mapeia em objetos do domínio, como estados e ações.

- **probabilistic_planning.rb**: arquivo que roda o programa. São recebidos dois parâmetros, os nomes do domínio e do algoritmo que será utilizado para resolver os problemas do domínio. É assumido que existe um diretório *results* criado, com dois subdiretórios, *tireworld* e *navigation_problem* (nome domínios suportados). O resultado dos algoritmos para cada problema em específico será armazenado dentro do diretório do seu respectivo domínio. Todos os problemas do domínio escolhido serão executados. A descrição deles pode ser encontrada nos subdiretórios do diretório *problems*.

- **domain/state.rb**: abstração dos estados. Possui atributos, como *stacked*, *visited* e *greedy_action* que auxiliam na elaboração da saída e cálculo de métricas para cada algoritmo.

- **domain/action.rb**: abstração das ações. O destino de cada ação é tratado como uma matriz, na qual cada linha representa um destino, a primeira coluna é a probabilidade e a segunda é o estado final.

- **domain/problem_definition.rb**: abstração do problema. É usada para fazer o cadastro dos estados e suas respectivas ações. Guarda informações gerais do problema, como estado inicial, meta e fator de desconto. Ordena os destinos de cada ação por ordem decrescente de probabilidade.

- **algorithms/procedures.rb**: módulo que possui funções comuns dos algoritmos implementados. É importante notar que levamos em consideração o fator de desconto do problema e a recompensa da ação do cálculo do *qValue*. A ação gulosa escolhida é aquela que possui maior *qValue*. Usamos uma distribuição de probabilidade uniforme para decidir qual o próximo estado escolhido.

- **algorithms/vi.rb**: implementação do algoritmo Iteração de Valor. A condição de parada é que todos os estados tenham convergido, ou seja, que o valor do residual de cada um deles seja menor que $\varepsilon = 10^{-7}$.

- **algorithms/rtdp.rb**: implementação do algoritmo RTDP. Apesar de explorar apenas os estados relevantes, não apresenta condição de parada. Portanto, aqui definimos um *timeout* de 1 minuto para o algoritmo, baseado no tempo de execução do LRTDP. Resolvemos os *dead ends* com um contador que verifica se a simulação (*trial*) permanece

no mesmo estado por mais de 30 vezes seguidas. Se isso acontecer, o *trial* é interrompido.

- **algorithms/lrtdp.rb**: implementação do LRTDP. Esse algoritmo é uma melhora do RTDP ao rotular os estados já resolvidos. O algoritmo para quando o estado inicial é resolvido. Para determinar se um estado está resolvido ou não, verificamos se seu residual é menor que $\varepsilon = 10^{-7}$. Resolvemos os *dead ends* com um contador que verifica se a simulação (*trial*) permanece no mesmo estado por mais de 30 vezes seguidas. Se isso acontecer, o *trial* é interrompido.

- **algorithms/ilao.rb**: implementação do algoritmo iLAO*. O critério de parada desse algoritmo é a convergência dos estados relevantes. Um estado convergiu se seu residual é menor que $\varepsilon = 10^{-7}$. Neste algoritmo resolvemos os *dead ends* colocando os estados visitados em uma pilha. Se o estado atual já está na pilha, o algoritmo simplesmente termina o *trial* e vai para o próximo.

5 Experimentos e Resultados

Rodamos os algoritmos implementados em dois domínios: Navigation e TireWorld, descritos nas seções 5.1 e 5.2 respectivamente. Durante a execução do algoritmo, extraímos diversas métricas do problema dentre elas: tempo de execução do algoritmo, número de estados expandidos, estados visitados e número de *trials*. Usamos essas métricas para analisar o desempenho de cada um dos algoritmos dos experimentos realizados.

5.1 Domínio Navigation

Neste domínio, temos uma grade e um robô deve ir de uma localização origem (estado inicial) da grade para uma localização destino (estado meta). O robô pode se movimentar para o norte, sul, leste ou oeste ou ainda escolher permanecer no mesmo lugar em que se encontra (ação noop).

Para cada estado em que o robô se encontra, ao executar uma determinada ação, pode existir uma probabilidade $p > 0$ do robô quebrar e se tornar impossível dele continuar sua trajetória e consequentemente alcançar a meta.

Para todo estado alcançado que não seja um estado meta, o robô recebe uma penalização.

A Figura 1 mostra um exemplo de uma instância do problema para este domínio. Na figura, as regiões em destaque (cor cinza) representa posições na grade onde, ao executar a ação que move o robô para o sul, existe uma

probabilidade do robô quebrar. A posição onde está a bandeira, representa o estado meta.

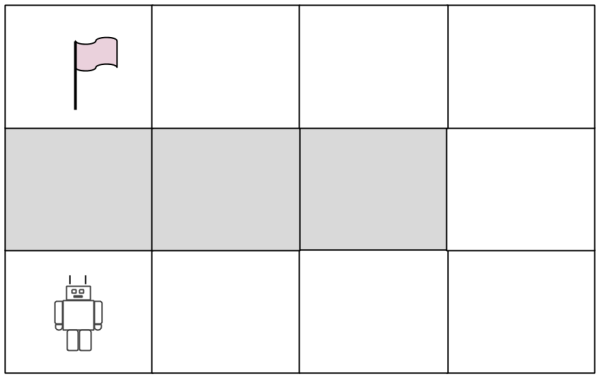

Figura 1: Exemplo de uma instância de problema do Domínio Navigation
Posições em destaque (cinza) representam posições da grade em que, ao executar a ação de mover para o sul e alcançar alguma dessas posições, existe uma probabilidade do robô quebrar.

5.1.1 Experimentos com o Domínio Navigation

Rodamos os algoritmos para doze diferentes problemas desse domínio. Cada problema variou o número de posições possíveis na grade, diferentes estados com probabilidade do robô quebrar, posição inicial e final.

5.1.2 Resultados para o Domínio Navigation

A Figura 2 exibe o tempo de execução de cada um dos algoritmos. Ao observarmos ela, é possível perceber que nos domínios menores, ou seja, domínios onde o espaço de estados possíveis é menor (por exemplo Nav-01, Nav-02, Nav-03 e Nav-04) o algoritmo $LRTDP$ converge mais rapidamente quando comparado ao $ILAO$. Porém, a partir do momento em lidamos com problemas maiores, o $ILAO$ passa a apresentar resultados melhores.

Como o $RTDP$ não possui uma condição de parada, nos experimentos definimos o *timeout* de execução em 1 minuto. Isso explica porque seus resultados estão muito acima dos demais algoritmos.

Os demais algoritmos ($ILAO$, $LRTDP$ e VI) que tem como critério de parada a convergência dos estados, o VI foi o que apresentou um desempenho pior. Ou seja, levou um tempo maior para convergir.

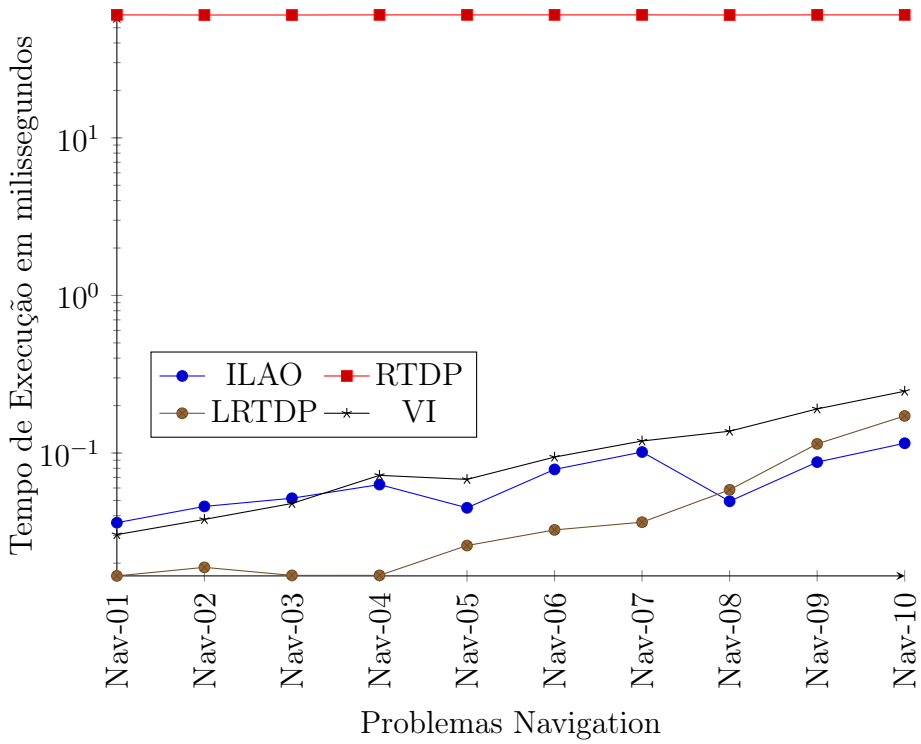

Figura 2: Tempo de Execução - Domínio Navigation

A Figura 3 apresenta o número de *trials* de cada um dos algoritmos ao longo de sua execução nas diferentes instâncias.

Ao observamos o algoritmo $RTDP$, podemos perceber que neste domínio, para os problemas menores, o número de *trials* foi maior que em problemas maiores (linha vermelha). Isso ocorre porque todas as instâncias foram executadas por um tempo limite padrão (*timeout*=1 minuto), como o *trial* das instâncias menores consome menos tempo, é natural que dado um tempo limite, instâncias menores consigam executar mais *trials*.

Neste domínio, que não possui política própria (uma política que quando executada a partir de um estado s alcança algum estado meta com probabilidade igual a 1), o algoritmo $ILAO^*$ executou um número maior de *trials* que VI e $LRTDP$.

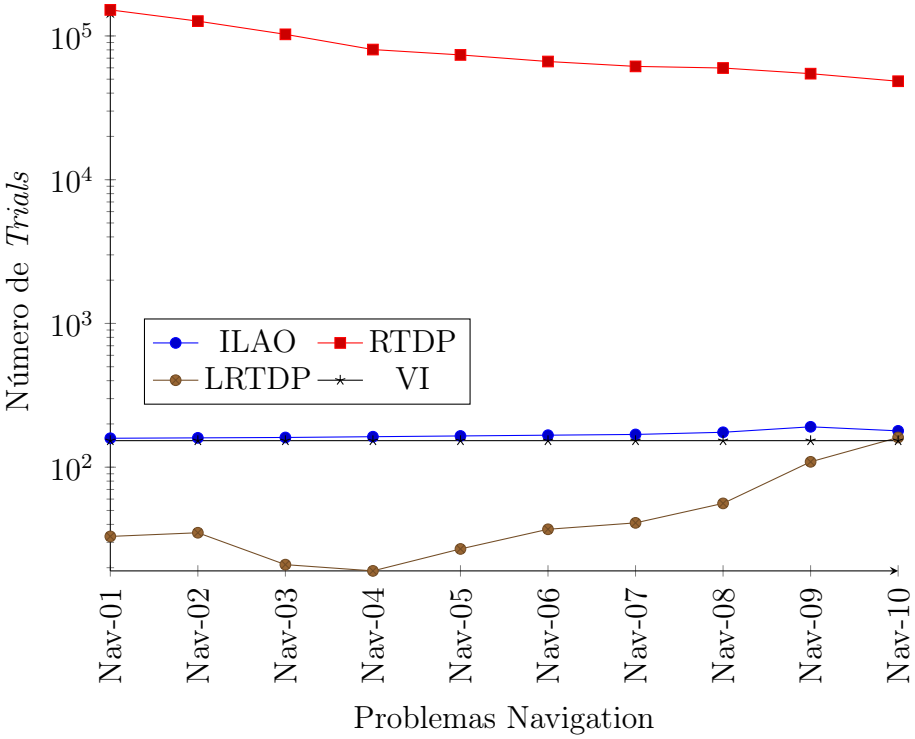

Figura 3: Número de *Trials* - Domínio Navigation

A Figura 4 mostra a quantidade de estados explorados pelos algoritmos. O algoritmo VI foi o que visitou o maior número de estados. Em nossos experimentos, este algoritmo sempre visita todos os estados do problema.

Ao compararmos os algoritmos $ILAO$ e $LRTDP$, percebemos que o número de estados visitados pelo $ILAO$ é maior que o número de estados visitados pelo $LRTDP$. Em alguns problemas, como por exemplo Nav-07, a diferença entre os estados visitados por esses algoritmos é bastante significativa.

Apesar do número de estados visitados pelo $ILAO$ em praticamente todos os problemas analisados ser maior, o tempo de convergência do $ILAO$ é menor. Isso porque, conforme citado anteriormente, o número de estados atualizados a cada iteração (e consequentemente visitados) pelo $ILAO$ é maior que o número de estados atualizados pelo $LRTDP$.

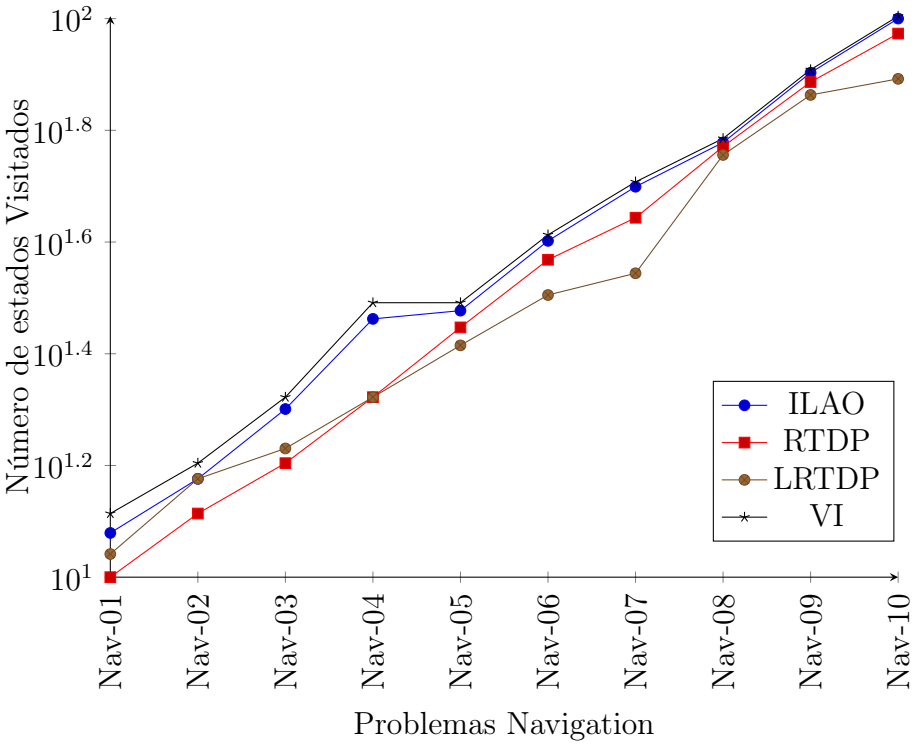

Figura 4: Número de estados visitados- Domínio Navigation

5.2 Domínio TireWorld

Assim como no domínio Navigation, este domínio consiste em um problema de navegação onde um carro deve se mover de uma localização inicial até uma localização meta, podendo passar, ao longo de sua trajetória, por diversas cidades. Durante seu percurso existe uma probabilidade do pneu do carro furar. Nestes casos, é possível trocar o pneu furado pelo step e, em algumas cidades, quando o carro não possui mais pneu reserva, ele pode adquirir um novo pneu.

A Figura 5 mostra o exemplo do ambiente para uma instância desse domínio. Na figura, cada ponto representa uma cidade, os pontos em destaque representam as cidades em que é possível repor o step enquanto os pontos brancos indicam as cidades onde não é possível repor o step. As setas entre os pontos indicam que existe um caminho entre as cidades. A posição com bandeira representa o estado meta.

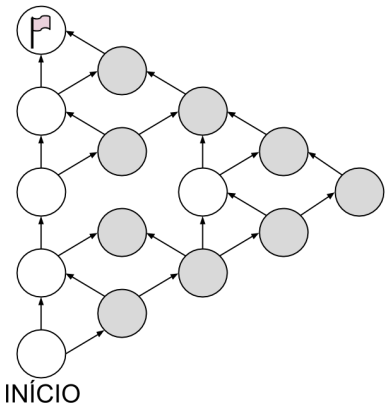

Figura 5

5.2.1 Experimentos com o Domínio TireWorld

Rodamos os algoritmos com dez diferentes configurações de problemas para este domínio. Para cada diferente problema, o número de estados e as probabilidade das ações variaram.

5.2.2 Resultados para o Domínio TireWorld

A Figura 6 exibe o tempo de execução de cada um dos algoritmos. Assim como no domínio Navigation, nos problemas do domínio Tireworld, o algoritmo VI foi o que apresentou o pior desempenho em termos de tempo de convergência. Neste domínio, em todos os problemas, o $ILAO$ foi o que teve o menor tempo de convergência.

Nestes experimentos também usamos *timeout* de um minuto para o algoritmo $RTDP$, o que pode explicar resultados muito acima dos demais algoritmos.

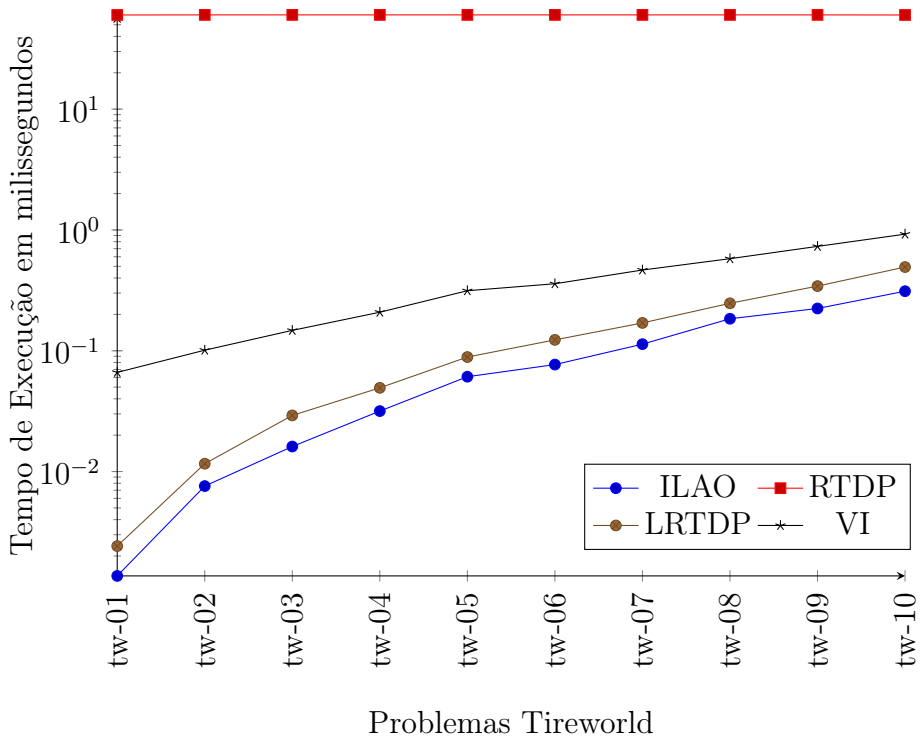

Figura 6: Tempo de Execução - Domínio Tireworld

A Figura 7 apresenta o número de *trials* de cada um dos algoritmos ao longo de sua execução nas diferentes instâncias.

Neste domínio, o algortimo $LRTP$ executou mais *trials* que o algoritmo $ILAO^*$, isso porque o algoritmo $ILAO$ atualiza mais estados a cada iteração.

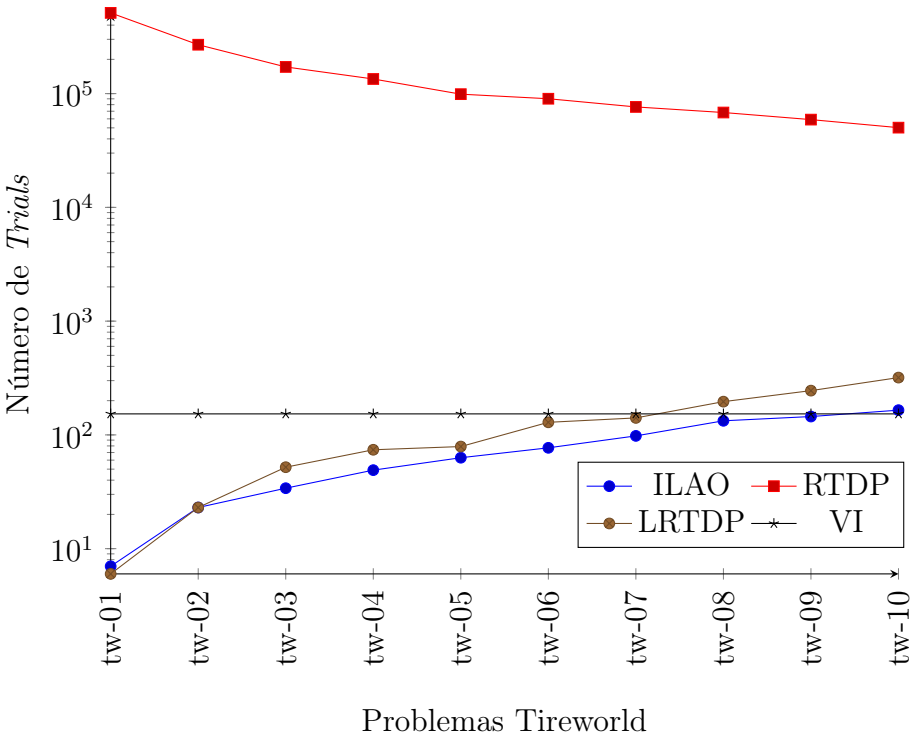

Figura 7: Número de *Trials* - Domínio Tireworld

A Figura 8 mostra a quantidade de estados visitados pelos algoritmos.

Assim como no domínio Navigation, o algoritmo VI visitou o maior número de estados em todos os problemas. Neste domínio, a diferença enter os estados visitados em cada um dos algoritmos foi muito pequena. Nos problemas pequenos (tw-01 e tw-02) o $RTDP$ foi o que visitou menos estados.

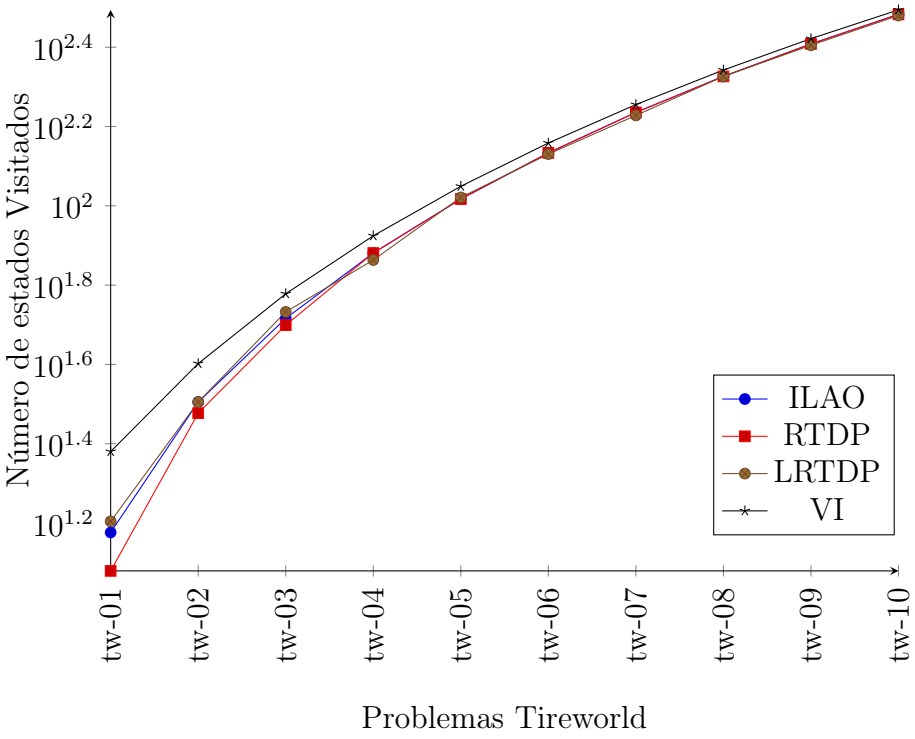

Figura 8: Número de estados visitados- Domínio Tireworld

6 Conclusões

Em geral, em nossos experimentos, o algoritmo $ILAO^*$ foi o que apresentou um melhor desempenho. Apenas em algumas instâncias menores, o $LRTDP$ teve uma performance melhor.

Apesar de apresentar um tempo de convergência menor, a quantidade de estados vistados pelo $ILAO^*$ foi maior. Em relação ao estados relevantes, podemos afirmar que ambos algoritmos tentam reduzir o número de estados visitados, pois usam políticas gulosas.

Um exemplo que ilustra isso, é o estado robot-at-x20-y04 (um estado irrelevante para os dois algoritmos). Nenhuma política ótima chega até ele. A principal diferença está em como tratar os possíveis efeitos não-determinísticos. O $LRTDP$ visita somente os estados mais prováveis, deixando muitas vezes de lado os estados que tem probabilidade baixa de serem alcançados, enquanto o algoritmo $ILAO^*$ não se limita em visitar apenas estados mais prováveis, mas todos os estados alcançáveis a partir de uma ação.

Referências

[Barto et al., 1995] Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition.

[Bonet, 2003] Bonet, B. (2003). Labeled rtdp: Improving the convergence of real-time dynamic programming. In *In Proc. ICAPS-03*, pages 12–21. AAAI Press.

[Hansen and Zilberstein, 2001] Hansen, E. A. and Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62.

[Russell et al., 1995] Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., and Edwards, D. D. (1995). *Artificial intelligence: a modern approach*, volume 74. Prentice hall Englewood Cliffs.