Plan Existence Verification as Symbolic Model Checking

Macilio da Silva Ferreira[1], **Maria Viviane Menezes**[1], **Leliane Nunes de Barros**[2]

[1] Universidade Federal do Ceará – Campus Quixadá

[2]Instituto de Matemática e Estatística – Universidade de São Paulo.

macilioferreira@alu.ufc.br, vivianemenezes@ufc.br, leliane@ime.usp.br

Abstract. *Automated Planning is the subarea of AI concerned with the generation of a plan of actions for an agent to achieve its goals. State-of-the-art planning algorithms are based on heuristic search. However, the inexistence of a plan can be a challenge for such planners, since they are not always able to discern the difficulty of finding a solution from its inexistence. The problem of plan existence verification, called* PLANEX*, is computationally hard. Thus, in 2016, the planning community held for the first time the Unsolvability International Planning Competition (UIPC), which aims to evaluate algorithms on the task of verifying plan existence. The aim of this paper is to propose a new algorithm to solve the* PLANEX *problem that is based on symbolic model checking approach. The proposed algorithm differs from others based on model checking in two points: (i) it is able to reason about the actions represented in PDDL (Planning Domain Description Language) and; (ii) it is based on the α-CTL logic, whose semantics takes into account the actions responsable for the state transitions. We also evaluate the proposed alorithm over the UIPC planning benchmark problems.*

Resumo. *Planejamento Automatizado é a subárea da IA que se preocupa com a elaboração de um plano de ações para que um agente alcance suas metas. Algoritmos eficientes de planejamento são baseados em busca heurística, no entanto, a inexistência de uma solução pode ser um desafio para estes planejadores uma vez que eles nem sempre conseguem discernir a dificuldade de encontrar um plano da sua inexistência. O problema de verificar a existência de um plano solução, chamado de* PLANEX*, é computacionalmente difícil. Assim, em 2016, a comunidade de planejamento automatizado organizou pela primeira vez a Unsolvabiity International Planning Competition (UIPC) voltada a testar o desempenho de algoritmos capazes de verificar a existência de um plano. Este artigo tem por objetivo propor um algoritmo para resolver o problema* PLANEX *que é baseado na abordagem formal de verificação simbólica de modelos. O algoritmo proposto difere de outros baseados em verificação de modelos da literatura em dois pontos: (i) é capaz de raciocinar sobre ações representadas em PDDL (Planning Domain Description Language) e; (ii) está baseado na lógica α-CTL, cuja semântica leva em consideração as ações responsáveis pelas transições entre estados. O algoritmo proposto também é avaliado utilizando os problemas da UIPC.*

1. Introdução

Na vida real, estamos sempre planejando nossas ações para que nossos objetivos sejam alcançados. *Planejamento Automatizado* é a subárea da Inteligência Artificial que estuda o processo de escolha e organização de ações para alcançar metas.

Um *domínio de planejamento* descreve a dinâmica do ambiente e pode ser representado explicitamente por meio de um grafo orientado e rotulado, em que: (i) os *nós* representam os estados do ambiente os quais são rotulados pelo conjunto de proposições que são verdadeiras nos estados; e (ii) as *arestas* são rotuladas pelas ações responsáveis pelas transições entre estados. Um *problema de planejamento* é definido por: um *domínio de planejamento*, um *estado inicial* s_0 e a descrição de uma *meta* φ. A solução para um *problema de planejamento* é um *plano*: uma sequência de ações que leva o agente do estado inicial para um estado que satisfaz a meta. Um *planejador* (Figura 1) é um algoritmo que recebe como entrada um problema de planejamento e devolve um *plano* ou *falha*, se não for possível encontrar um plano solução. Neste caso, dizemos que *não existe uma solução para o problema de planejamento*.

Figura 1. Esquema de um Planejador [Pereira and Barros 2008].

Domínios de planejamento realistas possuem um número muito grande de estados, o que impossibilita sua *representação explícita* como um grafo de transição de estados. Por esta razão, a comunidade de planejamento utiliza uma *representação implícita* dos domínios através de uma linguagem de descrição de ações. A linguagem PDDL (*Plannig Description Domain Language*) [McDermott et al. 1998] é a linguagem padrão de representação de domínios e problemas de planejamento. Em PDDL, as ações são definidas em termos de suas *pré-condições* (proposições que devem ser verdadeiras no estado em que a ação é executada) e *efeitos* (proposições que mudam no estado resultante da execução da ação).

A busca por um plano é uma tarefa computacionalmente difícil e o problema de decidir se existe ou não um plano solução para um problema de planejamento, denominado PLANEX PROBLEM, é PSPACE [Erol et al. 1995]. Dada esta complexidade, nos 40 anos de história da área de planejamento automatizado o foco da comunidade foi o desenvolvimento de algoritmos de planejamento baseados em busca heurística. No entanto, *problemas de planejamento sem solução* podem ser um grande desafio para os planejadores baseados em busca heurística, uma vez que esses algoritmos nem sempre são capazes de distinguir a dificuldade de encontrar uma solução da sua inexistência. Assim, em 2016, foi realizada pela primeira vez a *Competição Internacional de Planejamento para Problemas Sem Solução* (UIPC) [1] com o intuito de avaliar o desempenho dos algoritmos capazes de verificar a existência de um plano. Os algoritmos participantes desta competição podem ser divididos em três categorias: *baseados em busca heurística* (Aidos, h^{++}, *CLone* e *DE-PDB*), *baseados em planejamento como satisfação booleana*, (ReachLunch, iProverPlan, SimDominance, M+S) e *baseados em verificação simbólica de modelos* (SymPA).

[1] http://unsolve-ipc.eng.unimelb.edu.au/

A *Verificação Simbólica de Modelos* [Burch et al. 1992] é uma abordagem para verificar se um modelo formal de um sistema satisfaz uma fórmula lógica. Nesta abordagem, o conjunto de estados e a relação de transição são representados por *Diagramas de Decisão Binária* (BDDs) [Bryant 1986]. Essa abordagem têm sido usada para encontrar planos, uma área conhecida como *Planejamento como Verificação Simbólica de Modelos*. Algoritmos tradicionais de planejamento como verificação simbólica de modelos [Bertoli et al. 2001, Edelkamp and Helmert 2001, Cimatti et al. 2003, Giunchiglia and Traverso 2000] executam operações sobre triplas do grafo de transição de estados ⟨estado, ação, estado⟩; além disso, são baseados na lógica temporal CTL (Computation Tree Logic) [Clarke and Emerson 1982] que não considera as ações responsáveis pelas transições entre estados. [Fourman 2000] propôs um planejador baseado em verificação simbólica de modelos capaz de raciocinar sobre modelos implícitos determinísticos, isto é, usando uma linguaem de ações. [Pereira and Barros 2008] propôs o algoritmo `VACTL` de verificação de modelos baseado na lógica α-CTL, uma extensão da lógica CTL que considera as ações responsáveis pelas transições entre estados em sua semântica e operações de pré-imagem. No entanto, este algoritmo não é capaz de raciocinar sobre modelos com grande número de estados. [Menezes 2014, Menezes et al. 2014] propõem o algoritmo `VACTL-Sym` uma extensão do algoritmo de [Fourman 2000], baseada na lógica α-CTL, capaz de raciocinar sobre modelos implícitos, simbólicos e não-determinísticos.

Este artigo tem como objetivo propor um novo algoritmo para resolver o prolema de verificação de existência de plano, chamado de `MC-PLANEX`, que é baseado em verificação simbólica de modelos. O algoritmo `MC-PLANEX` estende o algoritmo `VACTL-Sym` e foi implementado com base em operações sobre BDDs. O restante do artigo está organizado da seguinte maneira. Na seção 2 introduzimos brevemente os conceitos de *Planejamento Automatizado*. A Seção 3 apresenta o Planejamento como Verificação Simbólica de Modelos. Em seguida são apresentados os resultados obtidos na análise experimental. Na seção 5 mostramos conclusões e discutimos trabalhos futuros.

2. Planejamento Automatizado

2.1. Representação de Domínios e Problemas de Planejamento

Um domínio de planejamento pode ser representado de maneira *explícita*, utilizando um sistema de transição de estados (como na Definição 1), ou *implícita*, utilizando uma linguagem de ações (como a linguagem `PDDL`).

Definição 1 *(***Representação Explícita de Domínios***) Dado um conjunto finito de proposições atômicas $\mathbb{P}$, descrevendo as características do ambiente, e um conjunto finito de ações $\mathbb{A}$, descrevendo as habilidades do agente no ambiente, um domínio de planejamento $\mathbb{D}$ pode ser representado explicitamente como um sistema de transição de estados $\mathbb{D} = \langle S, L, T \rangle$ em que [Pereira and Barros 2008]:*

- $\mathcal{S} \neq \emptyset$ *é um conjunto finito de estados.*
- $\mathcal{L} : S \mapsto 2^{\mathbb{P}}$ *é uma função de rotulação de estados, que descreve quais proposições atômicas são verdadeiras em um estado (Suposição do mundo fechado).*
- $\mathcal{T} : S \times \mathbb{A} \mapsto 2^{S}$ *é uma função de transição de estados em que cada transição é rotulada por uma ação $a \in \mathbb{A}$.*

Um *problema de planejamento* é definido em termos de uma representação do domínio de planejamento, como por exemplo na Definição 1, do estado inicial e da meta de planejamento. Formalmente, um *problema de planejamento* é definido por $\mathbb{D}$, um estado inicial s_0 e uma fórmula meta φ, como mostrado na Definição 2.

Definição 2 *(***Problema de Planejamento***) Um problema de planejamento é definido por uma tupla* $\mathbb{P} = \langle D, s_0, \varphi \rangle$ *em que:*

- $\mathcal{D}$ *é um domínio de planejamento.*
- $s_0 \in \mathbb{S}$ *é o estado inicial do problema, descreve a situação inicial do agente no ambiente.*
- φ *é a meta de planejamento, especificada por uma fórmula proposicional.*

Formalmente, uma ação a, sobre um conjunto de proposições $\mathbb{P}$, é definida por meio de suas pré-condições, efeitos positivos e efeitos negativos, conforme mostrado na Equação 1. Os efeitos da ação a são dados pela tupla $efeitos(a) = \langle efeitos^{+}(a), efeitos^{-}(a) \rangle$, sendo $efeitos^{+}(a)$ uma lista de proposições que se tornam verdadeiras após a execução da ação e $efeitos^{-}(a)$ uma lista de proposições que se tornam falsas após a ser executada.

$$a = \langle precond(a), efeitos(a) \rangle. \tag{1}$$

Em PDDL, o domínio de planejamento é representado pelo conjunto de proposições, descrevendo as propriedades do agente e do ambiente, e pelo conjunto de ações, descrevendo as habilidades do agente no ambiente.

2.2. Em busca de uma solução

A busca por um plano é realizada utilizando-se uma busca progressiva ou regressiva no espaço de estados induzido pelas ações. A busca progressiva gera estados sucessores, a partir de um estado inicial, até encontrar um estado meta; enquanto que, a busca regressiva gera estados predecessores, a partir de um conjunto de estados meta, até encontrar o estado inicial [Menezes 2014].

Na busca progressiva, dado um estado x e uma ação a, um estado sucessor é computado se a ação a é *aplicável* no estado x, ou seja, se todas as precondições da ação a são satisfeitas em x. Diante disso, simulamos a execução de uma ação determinística no estado x para computar o estado sucessor de x. Fazendo a suposição do mundo fechado na representação de estados, o estado sucessor é gerado a partir de x efetuando a remoção dos efeitos negativos de a no estado x e adicionando-se os efeitos positivos de a em x. Caso uma ação não seja aplicável em um estado, não é possível gerar um estado sucessor. As propriedades que valem em x e não são afetadas pela execução da ação continuam valendo no estado resultante (axioma de frame). Chamamos a operação de geração de um estado sucessor de um estado x por meio da ação a de *progressão de x por a*.

$$progr^{a}(x) = \begin{cases} (x \setminus efeitos^{-}(a)) \cup efeitos^{+}(a) & \text{se} \quad precond(a) \subseteq x \\ \emptyset & \text{caso contrário.} \end{cases} \tag{2}$$

A busca regressiva, por sua vez, inicia a partir da meta, gerando estados antecessores até que o estado inicial seja alcançado. Um conceito muito importante da busca

regressiva, que a difere da busca progressiva, é que na progressão temos a geração de um único *estado concreto*, uma vez que o estado inicial é completamente especificado (isto é, sabemos tudo o que é verdade e falso nesse estado). Já na busca regressiva são gerados *estados abstratos*, isto é, o conjunto de proposições gerado representa um *conjunto de estados*. Isto ocorre pois a meta descreve uma pequena quantidade do número total de proposições em $\mathbb{P}$. Por este motivo, dizemos que a busca progressiva é uma *busca no espaço de estados* e a busca regressiva é uma *busca no espaço de crenças*. Assim, a regressão de um estado abstrato x por uma ação a é realizada verificando-se se a ação a é *relevante* ao estado x, isto é, se a ação contribui para o estado x: (i) tornando uma das propriedades em x verdadeira e; não possuindo um efeito que negue uma propriedade de x. Após a certificação de que a ação a é relevante a x, o conjunto de estados predecessores de x pela ação a é gerado fazendo a remoção dos efeitos positivos de a em x e adicionando-se as precondições de a.

$$regr^a(x) = \begin{cases} (x \setminus efeitos^+(a)) \cup precond(a) & \text{se} \quad efeitos^+(a) \cap x \neq \emptyset \text{ e } efeitos^-(a) \cap x = \emptyset \\ \emptyset, & \text{caso contrário.} \end{cases} \tag{3}$$

3. Planejamento como Verificação Simbólica de Modelos

A *verificação de modelos* [Clarke and Emerson 1982] é um método formal utilizado para determinar se um modelo de um sistema $\mathcal{M}$ satisfaz ou não uma dada propriedade φ. Um verificador de modelos recebe como entrada o modelo $\mathcal{M}$ e a propriedade φ e devolve *sucesso* se o modelo satisfaz a propriedade ($\mathcal{M} \models \varphi$), ou um *contra-exemplo* se o modelo não satisfaz a propriedade ($\mathcal{M} \not\models \varphi$). O modelo do sistema é representado por uma estrutura de Kripke [Kripke 1963] e as propriedades são especificadas utilizando a lógica CTL [Clarke and Emerson 1982].

A *Verificação Simbólica de Modelos* [Burch et al. 1992] é uma abordagem amplamente utilizada para contornar o problema da explosão do espaço de estados, representando conjunto de estados e a relação de transição do modelo por meio de *Diagramas de Decisão Binária* (BDDs) [Bryant 1986]. Um BDD é um grafo acíclico, similar a uma árvore de decisão binária, em que os nós não terminais são rotulados com proposições e os nós terminais com 1 ou 0, indicando se a fórmula proposicional representada pelo BDD é verdadeira ou falsa, respectivamente.

A Verificação Simbólica de Modelos têm sido utilizada para obter planos, uma área conhecida como *Planejamento como Verificação Simbólica de Modelos*. A maioria das abordagens [Giunchiglia and Traverso 2000, Bertoli et al. 2001, Cimatti et al. 2003, Edelkamp and Helmert 2001] aplicam diretamente os conceitos da área de Verificação Simbólica de Modelos e incluem alguma forma de representar as ações do agente. Estas abordagens: (i) representam o domínio de planejamento de forma explícita por meio da representação e manipulação sobre a relação de transição (triplas $\langle$ estado,ação,estado$\rangle$ do grafo de transição de estados); (ii) utilizam a lógica CTL que não é capaz de representar as ações responsáveis pela transição entre estados e; (iii) utilizam algum método extra-lógica para extrair um plano candidato da estrutura de Kripke para só então aplicar operações de verificação de modelos sobre o plano.

O planejador *PropPlan* [Fourman 2000] é um algoritmo baseado em verificação simbólica de modelos que utiliza BDDs para representar conjuntos de estados e as ações

(Equação 1), definindo operações simbólicas de progressão e regressão para ações determinísticas (Definições 6 e 7). [Torralba 2016] estendeu a abordagem de [Fourman 2000] para realizar busca simbólica para problemas de planejamento com custo nas ações. A seguir, mostramos como representar simbolicamente um estado (Definição 3), um conjunto de estados (Definição 4) e uma ação (Definição 5) do domínio de planejamento, como fórmulas proposicionais.

Definição 3 *(Representação Simbólica de um Estado) Seja $\mathbb{P}$ um conjunto de proposições do domínio de planejamento, um estado s do domínio de planejamento pode ser representado por uma fórmula proposicional, denotada por $\xi(s)$:*

$$\xi(s) = \bigwedge_{p \in L(s)} p, \tag{4}$$

Definição 4 *(Representação Simbólica de Conjunto de Estados) Um conjunto de estados X de um domínio de planejamento pode ser representado por uma fórmula proposicional, denotada por $\xi(X)$:*

$$\xi(X) = \bigvee_{s \in X} \xi(s). \tag{5}$$

Definição 5 *(Representação Simbólica de Ações) Uma ação $a = \langle precond(a); efeitos(a)\rangle$ do domínio de planejamento pode ser representada por uma tupla de fórmulas $\langle \xi(precond(a)); \xi(efeitos(a))\rangle$ tal que:*

- *$\xi(precond(a))$ é uma conjunção de átomos representando as precondições de a,*

$$\xi(precond(a)) = \bigwedge_{p \in precond(a)} p \qquad e, \tag{6}$$

- *$\xi(efeitos(a))$ é uma conjunção de átomos representando os efeitos de a, ou seja,*

$$\xi(efeitos(a)) = \bigwedge_{q \in efeitos^{+}(a)} q \ \wedge \bigwedge_{r \in efeitos^{-}(a)} \neg r. \tag{7}$$

3.1. Progressão e Regressão Simbólica de Ações Determinísticas

Para computar tanto a progressão quanto a regressão simbólica por uma determinada ação a, é necessário definir o conjunto de todas proposições que estão envolvidas nos efeitos de a, denominado $modifica(a)$. As Definições 6 e 7 apresentam a definição formal para a progressão e regressão simbólica, respectivamente, através de uma ação a sobre um conjunto de estados.

Definição 6 *(Progressão simbólica) Seja uma ação $a = \langle \xi(precond(a)), \xi(efeitos(a)), modifica(a)\rangle$ e seja $\xi(X)$ a representação proposicional do conjunto de estados X, a progressão simbólica de X por meio de a é:*

$$sprogressao^{a}(X) = \exists modifica(a).(\xi(X) \wedge \xi(precond(a))) \wedge \xi(efeitos(a)).$$

Definição 7 *(Regressão simbólica) Seja a uma ação descrita por a tupla $\langle \xi(precond(a)), \xi(efeitos(a)), modifica(a)\rangle$ e seja $\xi(X)$ a representação proposicional do conjunto de estados X, a regressão simbólica de X por meio de a é [Menezes 2014]:*

$$sregressao^{a}(X) = \xi(precond(a)) \wedge \exists modifica(a).(\xi(efeitos(a)) \wedge \xi(X)).$$

Note que o quantificador existencial é aplicado sobre fórmulas proposicionais e se refere aos valores TRUE e FALSE das proposições nas fórmulas. Portanto essas definições usam lógica QBF (*Quantified Boolean Formulas*) [Staalmarck 2003], que descrevem formalmente as operações que faremos entre BDDs.

3.2. Algoritmo `MC-PLANEX`

O Algoritmo 1 mostra o pseudocódigo do verificador de existência de plano baseado em verificação simbólica de modelos, o qual denominamos de MC-PLANEX. Este algoritmo é uma adaptação do verificador simbólico de modelos `VACTL-Sym` [Menezes et al. 2014] para a tarefa de decidir se um problema de planejamento possui ou não solução. O algoritmo recebe como entrada um problema de planejamento dado por: (i) um conjunto de ações $\mathbb{A}^{DD}$ em que cada ação é representada simbolicamente por um BDD (Definição 5); (ii) um BDD representando o estado inicial s_0^{DD}, conforme a Definição 3 e; um BDD φ^{DD} representando a meta de planejamento. Para domínios de planejamento clássico, como os da UIPC, o algoritmo deve verificar, a partir de s_0, se a fórmula temporal "*existe algum caminho que, por alguma ação a, a meta φ é finalmente verdade*" é satisfeita no domínio. Na lógica α-CTL, expressamos essa fórmula como $\exists\diamond\varphi$, em que $\exists$ é um quantificador de caminhos, φ é a meta de planejamento, e $\diamond$ é o operador temporal finalmente [Pereira and Barros 2008].

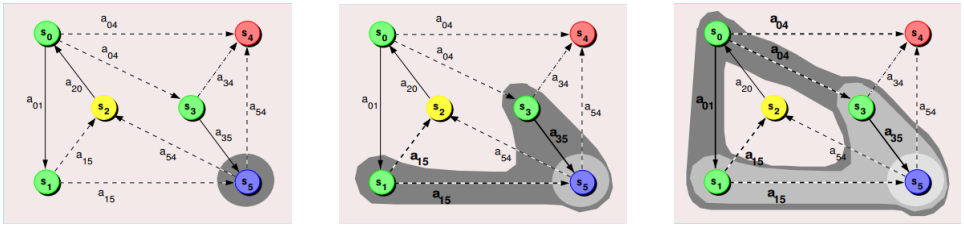
Algoritmo 1 MC-PlanEX($\mathbb{A}^{DD}$, s_0^{DD}, φ^{DD})

```
1: X^DD ← ⊥
2: Y^DD ← φ^DD
3: enquanto X^DD != Y^DD
4:     X^DD ← Y^DD
5:     Y^DD ← Y^DD ∨ sregressao(A^DD, Y^DD)
6:     se s_0^DD ∧ Y^DD != ⊥ então
7:         retorne 1                          ▷ existe um plano.
8: retorne 0                                  ▷ não existe um plano.
```

O algoritmo realiza uma busca regressiva no espaço de estados de crenças, iniciando a partir do conjunto de estados que satisfazem a meta. As variáveis X^{DD} e Y^{DD} são diagramas de decisão binária, utilizados para representar conjuntos de estados do domínio de planejamento. Na linha 1, a variável X^{DD} recebe valor *falso*, indicando um conjunto vazio de estados e a variável Y^{DD} recebe o BDD φ^{DD}, representando o conjunto de estados meta. Em cada iteração do laço das linhas 3-7, o BDD X^{DD} salva o conjunto de estados Y^{DD} (linha 4) e este conjunto (linha 5) é acrescido do conjunto de estados predecessores. Para computar os estados predecessores, a função *sregressao* recebe um conjunto de ações $\mathbb{A}^{DD}$, um conjunto de estados Y^{DD} e realiza o calculo da regressão (Definição 7) por meio de operações nos BDDs (operação *exists*). O laço é executado até que: (i) o estado inicial seja encontrado (linha 6) (i.e, existe um plano solução, fazendo com que o algoritmo retorne 1 na linha 7) ou; (ii) um ponto-fixo seja alcançado (garantido pela caracterização de ponto-fixo da lógica α-CTL [Pereira and Barros 2008]). No caso do laço encerrar ao atingir um ponto-fixo, significa que não foi possível encontrar o estado inicial e o problema de planejamento não possui solução, fazendo com que o algoritmo retorne 0 (linha 8).

Exemplo 1 *(Funcionamento do algoritmo* MC-PLANEX*) Considere um problema de planejamento em que o domínio é como o apresentado na Figura 2(a),* s_0 *é o estado inicial e* s_5 *é o estado meta. O algoritmo* MC-PLANEX *receberá uma representação implícita do domínio, dada pela representação das ações:* a_{01}, a_{04}, a_{15}, a_{20}, a_{34}, a_{35} *e* a_{54}*. O algoritmo realiza uma busca regressiva a partir do conjunto* Y^{DD} *de todos os estados que satisfazem a meta, neste caso o estado* s_5*. Em cada iteração, computa o conjunto de estados predecessores do conjunto* Y^{DD} *(utilizando a operação de regressão de ações) e os adiciona a* Y^{DD}*. Assim, temos:*

- *Primeira iteração:* $X^{DD} = s_5^{DD}$ *(BDD representando o estado* s_5*) e o retorno da função sregressao é* $s_1^{DD} \vee s_3^{DD}$*, BDD representando o conjunto de estados predecessores* $\{s_1, s_3\}$*. Assim, ao final da primeira iteração, temos que* $Y^{DD} = s_1^{DD} \vee s_3^{DD} \vee s_5^{DD}$*, como pode ser observado na Figura 2(b);*
- *Segunda iteração:* $X^{DD} = s_1^{DD} \vee s_3^{DD} \vee s_5^{DD}$ *e o retorno da função sregressao é* s_0^{DD}*. Assim, ao final da segunda iteração, temos que* $Y^{DD} = s_0^{DD} \vee s_1^{DD} \vee s_3^{DD} \vee s_5^{DD}$ *Figura 2(c). Como o estado* s_0 *foi alcançado, o algoritmo retorna 1, indicando que o problema de planejamento possui solução .*



(a) Antes da iteração 1 (b) Iteração 1 (c) Iteração 2

Figura 2. Busca regressiva para alcançar o estado inicial a partir do estado meta [Pereira 2007].

4. Análise Experimental

O algoritmo `MC-PlanEX` foi testado usando os domínios da UIPC-2016. Os objetivos desta análise experimental são: (E-I) avaliar o tempo de execução do planejador na tarefa de verificar se os problemas possuem ou não solução e; (E-II) comparar o desempenho do algoritmo proposto com relação aos sistemas participantes da UIPC, em termos da quantidade de problemas corretamente classificados como não solucionáveis.

Os domínios[2] utilizados nos experimentos foram: o domínio *sliding-tiles*, baseado no jogo *8-Puzzle*, que representa um tabuleiro de tamanho *nxn*, contendo $n-1$ peças numeradas e uma posição em branco; o domínio *bottleneck* que modela a tarefa de deslocamento de n pessoas em n localizações, sendo que apenas algumas posições estão livres e outras são intransitáveis; o domínio *pegsol-row5*, uma versão do jogo *resta-um* (*pegsol*) na qual as últimas 5 linhas são livres e as demais ocupadas por peças e; o domínio *cave-diving*, em que mergulhadores devem entrar em uma caverna subaquática e tirar fotos ou preparar o caminho para outros mergulhadores, deixando tanques cheios de ar na caverna.

Para a realização dos experimentos foi utilizado um computador com processador Intel Core i5, 8.0 GB de RAM e S.O Linux 16.04 LTS de 64 bits. No experimento (E-I) o

[2]Disponíveis em: bitbucket.org/planning-researchers/unsolve-ipc-2016/downloads/

tempo limite foi de 12 horas e a quantidade de memória foi limitada em 8GB. No experimento (E-II) o tempo limite foi de 30 minutos e a quantidade de memória permaneceu limitada em 8GB, respectivamente, que correspondem aos limites de tempo e memória estabelecidos na UIPC.

4.1. Desempenho sob as restrições da UIPC-2016

Esta seção mostra os resultados dos experimentos considerando um tempo limite de 12 horas. Os Gráficos 3, 4, 5 e 6 apresentam os resultados obtidos para os problemas dos domínios, respectivamente, *sliding-tiles*, *bottleneck*, *cave-diving* e *pegsol-row5*. Em cada gráfico, o eixo *x* mostra os problemas e o eixo *y* apresenta o tempo gasto pelo algoritmo na tarefa de decidir a existência de um plano. A linha horizontal destacada no gráfico expressa o limite de tempo de 30 minutos da UIPC-2016 e a letra *X* acima de um problema, especifica que o algoritmo não conseguiu decidir esse problema.

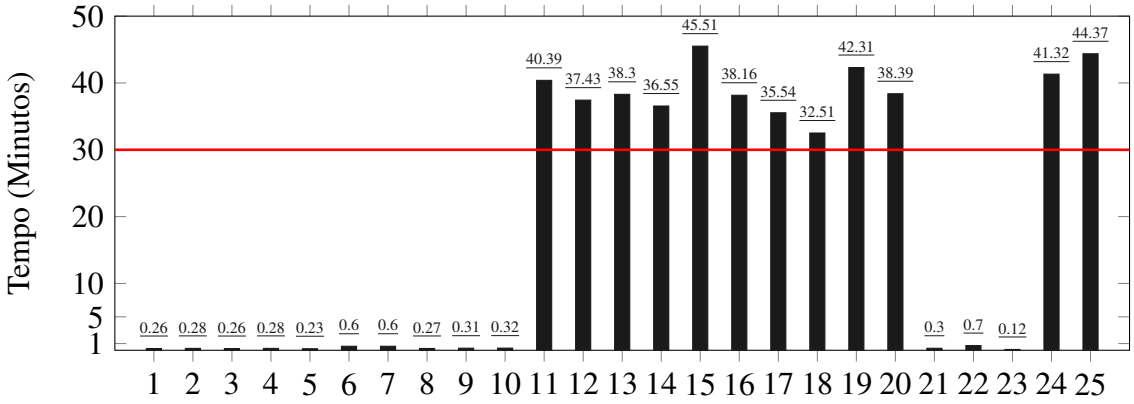


Figura 3. Resultado da execução do algoritmo sobre o domínio *sliding-tiles*

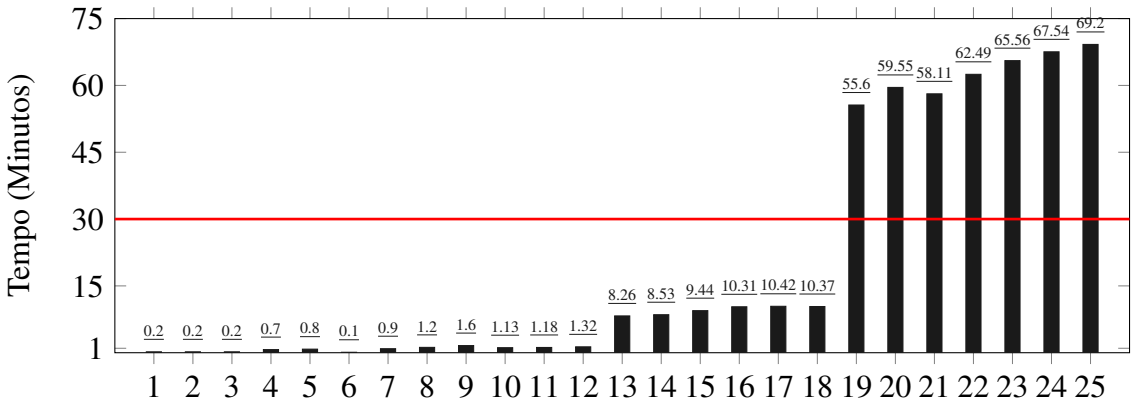


Figura 4. Resultado da execução do algoritmo sobre o domínio *bottleneck*

4.2. Análise Comparativa

Nesta seção mostramos uma análise comparativa entre o `MC-PLANEX` e os participantes da UIPC. Na competição, o objetivo é analisar a cobertura de problemas identificados corretamente como não solucionáveis. Os problemas solucionáveis somente foram introduzidos para evitar que um algoritmo pudesse devolver "*sem solução*" para todos os problemas.

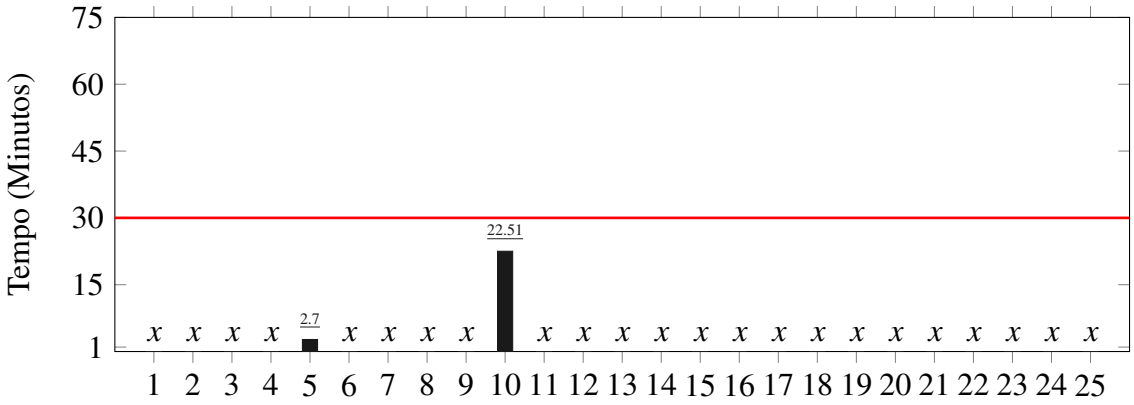


Figura 5. Resultado da execução do algoritmo sobre o domínio *cave-diving*

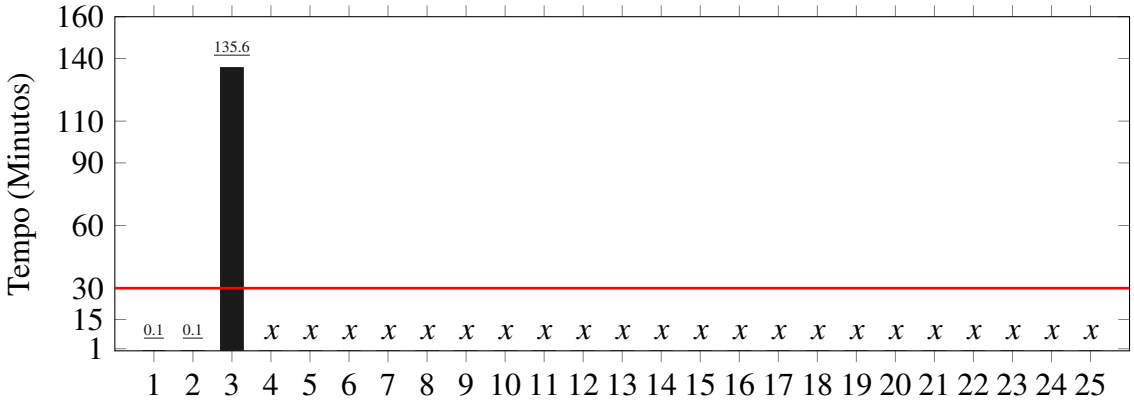


Figura 6. Resultado da execução do algoritmo sobre o domínio *pegsol-row5*

O algoritmo `MC-PLANEX`, com limite de tempo de 12 horas, conseguiu decidir corretamente como não solucionáveis 50 dos 85 problemas analisados, identificando corretamente todos os problemas do domínio *sliding-tiles* e do domínio *bottleneck*. No entanto, não obteve um bom desempenho nos domínios *pegsol-row5* e *cave-diving*, nos quais conseguiu identificar como não solucionáveis apenas 5 dos 40 problemas. Já com o limite de tempo utilizado na competição de 30 minutos, `MC-PLANEX` conseguiu decidir corretamente como não solucionáveis 32 dos 85 problemas. No entanto, nos domínios *sliding-tiles* e *bottleneck* o desempenho foi melhor do que nos domínios *pegsol-row5* e *cave-diving*.

O Gráfico 7 apresenta, para cada participante da UIPC, o número de problemas corretamente classificados como não-solucionáveis para os domínios *sliding-tiles*, *pegsol-row5*, *bottleneck* e *cave-diving* utilizados nesta análise. Observa-se que o `MC-PLANEX` para o domínio *sliding-tiles* conseguiu identificar corretamente a mesma quantidade de problemas que a maioria dos planejadores. Para o domínio *pegsol-row5* ele se aproximou da quantidade de problemas decididos pelo *SymPA* (segundo colocado na UIPC-2016) e ficou relativamente distante do Aidos (vencedor da UIPC-2016). O desempenho do `MC-PLANEX` sobre o domínio *bottleneck* é semelhante ao desempenho da maioria dos planejadores. No entanto, para o domínio *cave-diving* há uma grande diferença de desempenho em relação ao planejador Aidos, mas seu desempenho foi melhor do que o planejador IProver e se aproxima dos planejadores RLunch e h^{++}.

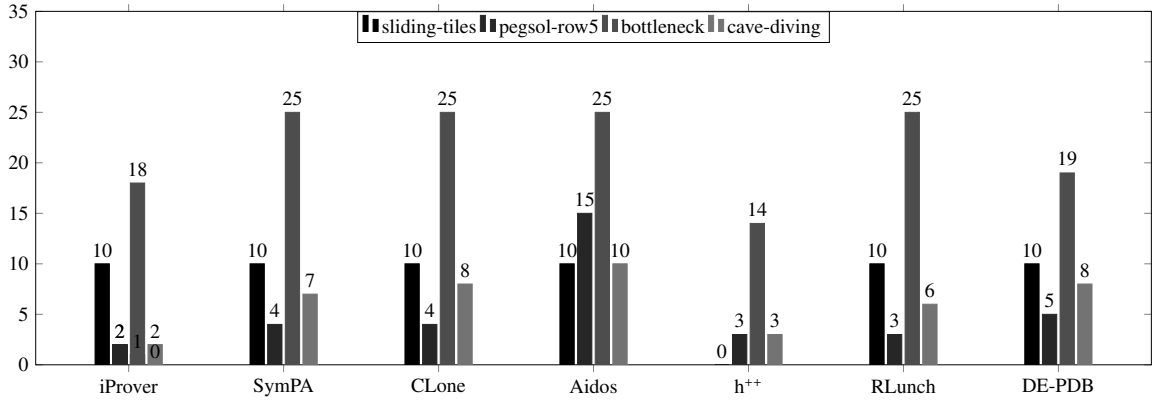


Figura 7. Comparativo do desempenho do `MC-PLANEX` com os participantes da UIPC.

A análise experimental também permitiu observar que é necessário adicionar, na descrição dos problemas, restrições sobre proposições mutualmente excludentes (*mutexes*), isto é, valorações de proposições que nunca podem ocorrer em um estado. Além disso, existem proposições que não são afetadas pelas ações (invariantes) e que poderiam ser descartadas da representação dos estados. De fato, notou-se que o algoritmo proposto obteve um desempenho ruim em domínios com uma grande quantidade de proposições invariantes.

5. Conclusões

Neste artigo, propomos e avaliamos o algoritmo de verificação de existência de plano, `MC-PLANEX`, baseado na lógica temporal α-CTL. No lado teórico, apresentamos os conceitos de *Planejamento Automatizado*, representação de um domínio de planejamento e como realizar a busca por um plano, e como a abordagem da verificação simbólica de modelos pode ser utilizada para verificar a existência de um plano. No lado prático, mostramos o algoritmo e os resultados obtidos com a análise experimental.

O desempenho do planejador `MC-PLANEX` na análise experimental foi satisfatório, visto que em dois domínios (*sliding-tiles* e *bottleneck*) conseguiu verificar corretamente todos os problemas sem solução. Contudo, para os outros dois domínios, *cave-diving* e *pegsol-row5*, `MC-PLANEX` não obteve um bom desempenho, mas apresentou um desempenho similar aos planejadores da competição. Essa análise permitiu apontar caminhos para a melhoria de desempenho do algoritmo: (i) realizar uma análise prévia do domínio para descartar as invariantes, (ii) adotar estratégias de redução do tamanho dos BDDs envolvidos na busca através da detecção de mutexes.

Como trabalhos futuros temos: a realização de experimentos para os outros 11 domínios *benchmarks* da UIPC-2016 e; a avaliação em domínios não-determinísticos; a melhoria da busca simbólica incluindo no algoritmo técnicas descritas por [Torralba 2016] como a busca bidirecional e a utilização do algoritmo A^*, adaptado para operações com BDDs.

Agradecimentos

Agradecemos à FAPESP (Projeto No. 2015/01587-0), à FUNCAP (projeto no BP2-01-07-00138.01.00/15) e ao CNPq pelo suporte financeiro.

Referências

[Bertoli et al. 2001] Bertoli, P., Cimatti, A., Pistore, M., Roveri, M., and Traverso, P. (2001). Mbp: a model based planner. In *Proc. of the IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*.

[Bryant 1986] Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691.

[Burch et al. 1992] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L.-J. (1992). Symbolic model checking: 1020 states and beyond. *Information and computation*, 98(2):142–170.

[Cimatti et al. 2003] Cimatti, A., Pistore, M., Roveri, M., and Traverso, P. (2003). Weak, strong, and strong cyclic planning via symbolic model checking. *AI*, 147(1-2):35–84.

[Clarke and Emerson 1982] Clarke, E. M. and Emerson, E. A. (1982). Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK. Springer-Verlag.

[Edelkamp and Helmert 2001] Edelkamp, S. and Helmert, M. (2001). MIPS: The model-checking integrated planning system. *AI Magazine*, 22(3):67.

[Erol et al. 1995] Erol, K., Nau, D. S., and Subrahmanian, V. S. (1995). Complexity, decidability and undecidability results for domain-independent planning. *Artificial intelligence*, 76(1):75–88.

[Fourman 2000] Fourman, M. P. (2000). Propositional planning. In *Proceedings of AIPS-00 Workshop on Model-Theoretic Approaches to Planning*, pages 10–17.

[Giunchiglia and Traverso 2000] Giunchiglia, F. and Traverso, P. (2000). Planning as model checking. *Recent Advances in AI Planning*, pages 1–20.

[Kripke 1963] Kripke, S. (1963). Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16:83–94.

[McDermott et al. 1998] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). Planning domain definition language.

[Menezes 2014] Menezes, M. V. (2014). *Mudanças em Problemas de Planejamento sem Solução*. PhD thesis, IME-USP.

[Menezes et al. 2014] Menezes, M. V., Barros, L. N., and Pereira, S. L. (2014). Symbolic regression for non deterministic actions. *Learning and Nonlinear Models*, pages 98–114.

[Pereira 2007] Pereira, S. L. (2007). *Planejamento sob incerteza para metas de alcancabilidade estendidas*. PhD thesis, Instituto de Matemática e Estatística da Universidade de São Paulo.

[Pereira and Barros 2008] Pereira, S. L. and Barros, L. N. (2008). A logic-based agent that plans for extended reachability goals. *AAMAS*, 16(3):327–344.

[Staalmarck 2003] Staalmarck, G. (2003). Quantified boolean formulas. In *Computer Aided Verification International Conference, Trento, Italy, 6-10, 1999*, page 23. Springer.

[Torralba 2016] Torralba, A. (2016). Sympa: Symbolic perimeter abstractions for proving unsolvability. *Unsolvability IPC: planner abstracts*, pages 8–11.