Silvio do Lago Pereira, Leliane Nunes de Barros

Eficiência num Planejador Abdutivo

Silvio do Lago Pereira, Leliane Nunes de Barros
Instituto de Matemática e Estatística da Universidade de São Paulo
Rua do Matão, 110 – 05508-900 – São Paulo – Brasil

Resumo Nesse artigo[1], modificamos um planejador abdutivo baseado em cálculo de eventos [17, 3] de modo a torná-lo equivalente a algoritmos selecionados da literatura de planejamento em Inteligência Artificial. Tal equivalência é estabelecida através de uma análise comparativa empírica (tempo e espaço de busca) entre quatro planejadores que diferem entre si quanto à abordagem empregada (lógica abdutiva com cálculo de eventos *versus* algorítmica com STRIPS) e quanto ao método de busca (sistemático *versus* não-sistemático). Com isso mostramos que, usando um formalismo puramente lógico, é possível obter sistemas de planejamento cuja eficiência seja equiparável àquela observada nos sistemas produzidos através da abordagem algorítmica.

1 Introdução

Um agente robótico ou de *software* deve ser capaz de planejar suas ações com antecedência. Tal habilidade é essencial ao comportamento inteligente e sua implementação é extremamente importante em aplicações práticas, e.g. robótica, manufatura, logística, missões espaciais, etc. De fato, planejar é uma tarefa bastante comum no nosso dia-a-dia, já que o correto seqüenciamento de passos é, geralmente, necessário para que objetivos específicos sejam atingidos.

Um grande número de algoritmos de planejamento foram propostos nos últimos trinta anos na área de Inteligência Artificial. Aqueles provados corretos possuem grandes limitações, em particular, quanto à representação de ações e, conseqüentemente, não podem ser usados para resolver problemas no mundo real. Por outro lado, os planejadores práticos, capazes de resolver problemas grandes, em geral, foram construídos de maneira *ad-hoc*, sendo difícil explicar porque funcionam ou porque seu comportamento pode ser considerado inteligente.

Assim, acredita-se que através do uso de lógica formal é possível construir planejadores corretos, fundamentados em princípios bem conhecidos, que podem ser facilmente validados, mantidos e modificados. Resta investigar se é possível construir planejadores baseados em lógica que sejam considerados "práticos" e capazes de resolver problemas do mundo real.

Green [8] foi o primeiro a implementar um sistema de planejamento dentro de uma abordagem lógica. Entretanto, embora seu sistema tenha sido muito admirado do ponto de vista teórico, na prática, ele se mostrou bastante ineficiente. Tal ineficiência é devida, sobretudo, à necessidade de se manter um grande número de axiomas para estabelecer que propriedades e relações persistem no mundo, após a execução de uma ação (*problema do quadro*). Recentemente, entretanto, *Shanahan* [17] publicou um artigo em que a abordagem lógica é resgatada. Nesse artigo, ele mostra que, usando *cálculo de eventos circunscritivo* como formalismo para

[1]Financiado pela CAPES e CNPq - projeto MAPPEL

raciocinar sobre ações e seus efeitos, e *programação lógica abdutiva* como técnica de prova de teoremas, é possível reproduzir a computação efetuada por um algoritmo de planejamento.

Em [17], *Shanahan* estabelece uma correspondência entre o AECP e o POP, baseando-se na simples inspeção de código. Em [3], *Barros & Santos* fazem uma análise no nível do conhecimento, estabelecendo a correspondência entre o AECP e uma biblioteca de métodos de planejamento. Nesse artigo implementamos alguns desses métodos, tornando o planejador abdutivo equivalente a algoritmos selecionados da literatura de planejamento em IA. Essa equivalência foi estabelecida através de uma análise empírica do comportamento de quatro planejadores proposicionais: ABP, SABP, POP e SNLP. Com isso mostramos que, usando um formalismo lógico, é possível obter planejadores cuja eficiência seja equiparável àquela observada na abordagem algorítmica. Também mostramos que mudanças devem ser feitas no planejador abdutivo para adaptá-lo às diferentes características de domínios de planejamento.

1.1 A Tarefa de Planejamento

Definição 1. *Dada uma descrição das ações que um agente pode executar, o estado inicial do mundo e os objetivos desse agente, a tarefa de planejamento consiste em determinar uma seqüência de ações (plano) que, quando executada num mundo que satisfaz a descrição do estado inicial, leva a um estado final onde os objetivos especificados são satisfeitos.* □

Em geral, planejadores clássicos descrevem ações através de operadores STRIPS [7], que estabelecem suas precondições e efeitos. Por exemplo, $Op(act\colon Andar(A,B), pre\colon\{Em(A)\}, add\colon \{Em(B)\}, del:\{Em(A)\})$ descreve a ação de andar de um local a outro. Ademais, em planejadores de ordem parcial [18], planos são representados por triplas da forma $\langle \mathcal{P}, \mathcal{O}, \mathcal{V}\rangle$, onde $\mathcal{P}$ é o conjunto de *passos*, $\mathcal{O}$ é o conjunto de restrições de *ordem* temporal e $\mathcal{V}$ é o conjunto de *vínculos causais*. As restrições de ordem impõem uma ordem parcial sobre os passos do plano e os vínculos estabelecem o propósito de cada um deles. Vínculos são estruturas da forma $\alpha_p \xrightarrow{\phi} \alpha_c$, onde α_p é um passo cujo efeito ϕ é precondição do passo α_c. Um passo $\alpha_a \in \mathcal{P}$ ***ameaça*** um vínculo $\alpha_p \xrightarrow{\phi} \alpha_c \in \mathcal{V}$ se $\mathcal{O} \cup \{\alpha_p \prec \alpha_a \prec \alpha_c\}$ é consistente e $\phi \in del(\alpha_a)$. Para evitar ameaças, protegemos os vínculos adicionando restrições de ordem ao plano: se um passo α_a ameaça um vínculo $\alpha_p \xrightarrow{\phi} \alpha_c$, então ele deve ser *antecipado* ou *postergado*, i.e. executado antes de α_p ou após α_c, respectivamente. Se passos que adicionam uma precondição já atingida por outro passo também são considerados como ameaça, o planejador é *sistemático* [13].

2 Raciocínio Abdutivo

Abdução, originalmente introduzida por *Peirce* [15], é uma regra de inferência que estende a dedução, possibilitando raciocínio hipotético: se observamos o fato β, e sabemos que $\alpha \rightarrow \beta$, então podemos adotar a hipótese α como uma possível explicação para β.

Definição 2. *Sejam Δ um conjunto de sentenças descrevendo um domínio e Γ_0 uma sentença descrevendo uma observação nesse domínio. A abdução consiste em encontrar um conjunto de sentenças Σ, explicação abdutiva para Γ_0, tal que $\Delta \cup \Sigma$ seja consistente e $\Delta \cup \Sigma \models \Gamma_0$.* □

Evidentemente, abdução é uma inferência fraca já que para um mesmo fato podem existir diversas explicações. Por exemplo, seja $\Gamma_0 := \{Enchente\}$ uma observação no domínio $\Delta :=$

$\{Verao \rightarrow Chuva, Umidade \wedge Calor \rightarrow Chuva, Chuva \rightarrow Enchente\}$. Então, $\Sigma_1 := \{Verao\}$, $\Sigma_2 := \{Umidade, Calor\}$, $\Sigma_3 := \{Chuva\}$ e $\Sigma_4 := \{Enchente\}$ são explicações abdutivas plausíveis para Γ_0. Para evitar explicações arbitrárias, tais como Σ_3 e Σ_4, os fatos numa explicação abdutiva são restringidos a pertencer a um conjunto básico de causas primitivas, i.e. *abdutíveis* [9].

2.1 O Mecanismo Abdutivo em Programação Lógica

Sejam Δ um programa lógico e Γ_0 uma meta. Uma SLD-*refutação* [1] de Γ_0 a partir de Δ é uma seqüência finita de metas $\Gamma_0, \ldots, \Gamma_n$, onde Γ_n é a cláusula vazia e cada Γ_{i+1} é uma resolvente de Γ_i com uma cláusula em Δ. Seja $\Gamma_0 := \leftarrow \gamma_1, \ldots, \gamma_k$ e suponha que não haja uma tal refutação para Γ_0. Então, da completude desse procedimento [1], segue que $\Delta \not\models \gamma_1 \wedge \ldots \wedge \gamma_k$. Suponha, então, que desejamos determinar de que forma Δ pode ser estendido por um conjunto de cláusulas Σ tal que $\Delta \cup \Sigma \models \gamma_1 \wedge \ldots \wedge \gamma_k$. Claramente, um tal Σ é uma explicação abdutiva para $\gamma_1 \wedge \ldots \wedge \gamma_k$ a partir de Δ . Para encontrar essa explicação, alteramos o procedimento de refutação do seguinte modo: Seja γ_i o literal selecionado de Γ_i que não resolve com nenhuma cláusula em Δ. Então, se γ_i é *abdutível*, incluímos em Σ uma cláusula unitária que resolva com ele e continuamos a refutação com Γ_i igual a Γ_{i+1}, exceto pelo literal γ_i, que é removido. O conjunto cumulativo de hipóteses abdutivas Σ é denominado *resíduo abdutivo* [9].

2.2 Abdução e Negação por Falha

Em *Programação Lógica*, a negação é implementada através de *negação por falha*, que estabelece que podemos inferir $\neg\lambda$ de um programa se não há uma SLD-refutação para a meta $\leftarrow \lambda$ (*prova em modo negativo*). A negação por falha supõe a *hipótese do mundo fechado* [16] e, portanto, o *completamento* do programa [5]. Estendendo-se SLD-refutação com negação por falha e abdução obtemos o procedimento SLDNFA-*refutação* [9]. Note que, por serem operações não-monotônicas, abdução e negação por falha interferem entre si. Na negação por falha apenas as conseqüências lógicas do programa mais o resíduo abdutivo já computado devem ser consideradas e, portanto, a abdução deve ser desabilitada durante as provas em modo negativo. Por outro lado, quando novas hipóteses são adicionadas ao resíduo abdutivo, conclusões negativas previamente provadas podem se tornar falsas, o que deve ser evitado. Assim, para manter a corretude da negação por falha, somente hipóteses consistentes com as negações já provadas (*resíduo negativo*) podem ser adicionadas ao resíduo abdutivo.

2.3 O Meta-interpretador Abdutivo

O meta-interpretador abp (figura 1) implementa SLDNFA-*refutação*. Seja $\mathcal{P}$ um programa, $\mathcal{A}$ um conjunto de abdutíveis e G uma meta. A chamada $abp(G, [\,], A, [\,], N)$ produz um resíduo abdutivo $A \subseteq \mathcal{A}$ e um resíduo negativo N tais que $\mathcal{P} \cup A$ é consistente e $Comp(\mathcal{P} \cup A) \models \{G\} \cup N$.

3 Cálculo de Eventos Circunscritivo

Cálculo de eventos, introduzido por *Kowalski & Sergot* [12], é um formalismo lógico para raciocínio sobre ações e efeitos. Trata-se de uma linguagem de primeira ordem tipada, cujas primitivas ontológicas incluem *eventos*, *fluentes* e *instantes de tempo*. Informalmente, a idéia básica do cálculo de eventos é estabelecer que um fluente *vale* num determinado instante do

```
abp([],R,R,N,N).                                                                    (1)
abp([G1|Gs],Ri,Ro,Ni,No) :- abducible(G1), nafs(Ni,[G1|Ri]), abp(Gs,[G1|Ri],Ro,Ni,No).   (2)
abp([G1|Gs1],Ri,Ro,Ni,No) :- axiom(G1,Gs2), append(Gs2,Gs1,Gs3), abp(Gs3,Ri,Ro,Ni,No).   (3)
abp([not(G1)|Gs],Ri,Ro,Ni,No) :- naf([G1],Ri), abp(Gs,Ri,Ro,[[G1]|Ni],No).       (4)

nafs([],R).                                                                         (5)
nafs([N|Ns],R) :- naf(N,R), nafs(Ns,R).                                             (6)

naf([G1|_],R) :- not resolve(G1,R,_).                                               (7)
naf([G1|Gs1],R) :- findall(Gs3,(resolve(G1,R,Gs2),append(Gs2,Gs1,Gs3)),Gss), nafs(Gss,R). (8)

resolve(G1,R,[]) :- member(G1,R).                                                   (9)
resolve(G1,R,Gs) :- axiom(G1,Gs).                                                  (10)
```

Figura 1: *O meta-interpretador abdutivo*

tempo se ele vale *inicialmente*, ou foi *iniciado* pela ocorrência de uma ação em algum instante anterior, e não foi *terminado* pela ocorrência de uma outra ação nesse meio tempo.

$$HoldsAt(f,t) \leftarrow Initially_P(f) \ \wedge \ \neg Clipped(0,f,t) \quad \text{(EC1)}$$
$$HoldsAt(f,t) \leftarrow Happens(a,t') \ \wedge \ Initiates(a,f,t') \ \wedge \ t' \prec t \ \wedge \ \neg Clipped(t',f,t) \quad \text{(EC2)}$$
$$Clipped(t_1,f,t_2) \leftrightarrow \exists a,t'[Happens(a,t') \ \wedge \ t_1 \prec t' \ \wedge \ t' \prec t_2 \ \wedge Terminates(a,f,t')] \quad \text{(EC3)}$$

Além dos axiomas EC, que capturam a persistência dos fluentes, precisamos de axiomas que descrevam que fluentes valem inicialmente e quais os efeitos das ações num domínio específico. Então, dada uma *narrativa* de ocorrências de ações, podemos determinar a validade de um fluente em qualquer instante do tempo. Por exemplo, dados os axiomas dependentes de domínio DD a seguir e a narrativa $\{Happens(Andar(A,B),t_1), Happens(Andar(B,C),t_2), t_0 \prec t_1, t_1 \prec t_2, t_2 \prec t_3\}$, podemos concluir $HoldsAt(Em(C),t_3)$.

$$Initially(Em(A)) \quad \text{(DD1)}$$
$$Initiates(Andar(x,y),Em(y),t) \leftarrow HoldsAt(Em(x),t) \ \wedge \ x \neq y \quad \text{(DD2)}$$
$$Terminates(Andar(x,y),Em(x),t) \leftarrow HoldsAt(Em(x),t) \ \wedge \ x \neq y \quad \text{(DD3)}$$

3.1 Circunscrição: uma solução para o problema do quadro

O *problema do quadro* [14] é tratado no cálculo de eventos através de *circunscrição*, que minimiza a extensão de um predicado. Por exemplo, sejam $\Sigma := Happens(A_1,T_1) \wedge Happens(A_2,T_2)$ e CIRC$[\Sigma; Happens]$ a circunscrição de Σ minimizando a extensão do predicado $Happens$, i.e. tornando-o verdade apenas para as tuplas que Σ o força a ser. Então, temos CIRC$[\Sigma; Happens] \models \forall a,t[Happens(a,t) \leftrightarrow (a = A_1 \wedge t = T_1) \vee (a = A_2 \wedge t = T_2)]$. Note que, circunscrevendo $Happens$, assumimos que não há ocorrências de eventos inesperados e, circunscrevendo $Initiates$ e $Terminates$, que as ações não têm efeitos inesperados.

3.2 Planejamento abdutivo com cálculo de eventos

Conforme *Eshghi* [6], um plano pode ser visto como uma explicação de como um estado meta pode ser atingido, a partir de um determinado estado inicial do mundo. Na verdade, planejamento pode ser *naturalmente* visto como uma tarefa abdutiva quando o cálculo de eventos circunscritivo é usado como formalismo para raciocínio sobre ações e efeitos. Dentro dessa concepção, a *tarefa de planejamento abdutivo* pode ser definida como segue.

Definição 3. *Sejam Δ uma conjunção de fórmulas descrevendo as ações de um domínio, Σ uma conjunção de fórmulas descrevendo que fluentes valem no estado inicial, Ω uma*

conjunção dos axiomas de unicidade-de-nomes, EC *a conjunção dos axiomas do cálculo de eventos e* Γ *uma conjunção de fórmulas descrevendo os objetivos a serem atingidos. A tarefa de planejamento abdutivo consiste em encontrar uma narrativa* Π *tal que*

(1) CIRC$[\Delta; Initiates, Terminates] \wedge$CIRC$[\Sigma \wedge \Pi; Happens] \wedge \Omega \wedge$EC *é consistente, e*
(2) CIRC$[\Delta; Initiates, Terminates] \wedge$CIRC$[\Sigma \wedge \Pi; Happens] \wedge \Omega \wedge$EC$\models \Gamma$. □

3.3 O planejador AECP *– Abductive Event Calculus Planner*

Como *Shanahan* [17] mostrou, o meta-interpretador abdutivo pode ser especializado para o cálculo de eventos através da compilação dos axiomas EC em metacláusulas. Por exemplo, o axioma (EC2) pode ser compilado na metacláusula apresentada na figura 2, onde o predicado $before$ denota restrições de ordem temporal. A vantagem dessa compilação é que ela proporciona um grau extra de controle que nos permite melhorar a eficiência (ajustando a ordem em que as submetas são resolvidas), bem como dar tratamento especial a predicados abdutíveis. Note que, no contexto de informação incompleta sobre um predicado, não podemos assumir o seu completamento e, portanto, não podemos usar negação por falha para provar literais negativos de tal predicado. Por exemplo, no planejamento de ordem parcial, a informação incompleta sobre o predicado $before$ é que nos permite representar conjuntos de ações apenas parcialmente ordenados. Assim, para provar $not(before(X, Y))$ o meta-interpretador tenta provar $before(Y, X)$, adicionando-o ao resíduo e verificando se esse permanece consistente.

```
abp([holds_at(F,T2)|Gi],Ai,Ao,Ni,No) :-    % seleciona uma meta F a ser atingida
  axiom(initiates(A,F,T1),G),               % escolhe ação A que produz F como efeito
  resolve(happens(A,T1),Ai,A1),             % adiciona uma instância A' dessa ação ao plano
  resolve(before(T1,T2),A1,A2),             % adiciona uma restrição de ordem ao plano
  add_neg([clipped(T1,F,T2)],Ni,N),         % adiciona um vínculo causal ao plano
  nafs(N,A2,A3),                            % trata os conflitos de interação de ações
  append(Gi,G,Go),                          % adiciona as precondições de A' como novas metas
  abp(Go,A3,Ao,N,No).                       % satisfaz as demais metas...
```

Figura 2: *Correspondência entre o* AECP *e o* POP

O AECP [17] nada mais é que o resultado da compilação dos axiomas EC em cláusulas do meta-interpretador abdutivo. Dado um programa descrevendo a situação inicial ($initially$) e as ações do domínio ($initiates$ e $terminates$), fornecemos uma lista de metas ($holds\_at$) ao AECP e ele nos devolve um resíduo ($happens$ e $before$) que representa um plano *parcialmente ordenado*. Note que esse plano contém apenas as restrições de ordem estritamente necessárias para manter as relações causais e para resolver possíveis conflitos entre as ações abduzidas; entretanto, qualquer linearização (*ordenação topológica*) dele, quando executada a partir do estado inicial, leva a um estado final em que as metas do planejamento são atingidas.

3.4 Correspondência entre o AECP *e o* POP

Existe uma clara correspondência (figura 2) entre os passos de planejamento num algoritmo de planejamento de ordem parcial (POP) e aqueles realizados pelo meta-interpretador abdutivo especializado para o cálculo de eventos. Na verdade, essa correspondência existe também em termos da representação de plano usada nesses dois sistemas: o conjunto $\mathcal{P} \cup \mathcal{O}$ corresponde ao resíduo positivo, consistindo de fatos $happens$ e $before$ abduzidos, e o conjunto $\mathcal{V}$ corresponde ao resíduo negativo, consistindo de fatos $clipped$ provados por negação por

falha, cuja consistência deve ser preservada pelo meta-interpretador. Manter a consistência do resíduo negativo no meta-interpretador equivale a proteger o estabelecimento de submetas (ou tratar ameaças) no planejamento clássico. Outro ponto importante é que qualquer descrição de ações em STRIPS pode ser convertida, em tempo polinomial, para uma representação em cálculo de eventos correspondente.

4 Análise de Eficiência do Planejador Abdutivo

Em [3], *Barros & Santos* fazem uma análise no nível do conhecimento que estabelece a correspondência entre o AECP e uma biblioteca de métodos de planejamento. Com base nessa correspondência, apresentamos uma análise comparativa entre planejamento abdutivo e planejamento clássico de ordem parcial. Para tanto, avaliamos empiricamente o comportamento de quatro planejadores proposicionais: (1) o ABP, versão proposicional do AECP; (2) o SABP, versão sistemática do ABP; (3) o POP [18], planejador de ordem parcial clássico; e, (4) o SNLP [13], versão sistemática do POP. Para garantir a consistência da análise, todos os planejadores foram implementados em SWI-PROLOG, compartilhando estruturas de dados e procedimentos sempre que possível (sem alterar a natureza dos planejadores).

4.1 O Planejador ABP*: uma versão proposicional especializada do* AECP

O ABP resulta de uma série de modificações que foram feitas no AECP. Apesar de algumas delas implicarem em menos expressividade, elas foram necessárias para que a comparação fosse precisa e realmente justa. As principais modificações foram motivadas pela observação de suposições que são geralmente feitas em planejamento clássico (e.g. POP e SNLP), a saber: (*i*) *tempo atômico*, (*ii*) *efeitos determinísticos* e (*iii*) *onisciência* [18]. Da suposição (*i*), segue que as ocorrências de ações são instantâneas e que, portanto, precisamos apenas da versão binária de $happens$. De (*ii*), segue que não precisamos do predicado $releases$, que serve justamente para implementar não-determinismo. E, finalmente, de (*iii*), mais a *hipótese do mundo fechado* [16], segue que não precisamos das versões negativas dos axiomas EC.

Representação de ações Para permitir o tratamento no metanível, as cláusulas do programa-objeto no AECP são representadas pelo predicado $axiom(H, B)$, onde H é a cabeça da cláusula e B é o seu corpo. Na representação STRIPS o primeiro argumento do predicado $op/4$ é o identificador de ação, enquanto na representação EC, o primeiro argumento do predicado $axiom/2$ é $initiates$ ou $terminates$. Como o PROLOG indexa as cláusulas na base de conhecimento usando esse primeiro argumento como chave, uma consulta com o predicado $op/4$ é respondida, praticamente, em tempo constante; já uma consulta com o predicado $axiom/2$ consome tempo proporcional ao número de cláusulas para o predicado $axiom/2$ existentes na base. Então, para uma melhor correspondência, as cláusulas da forma $axiom(\rho(\ldots), B)$ são mantidas no metanível como $\rho(\ldots, B)$. Ademais, deixamos implícito que os predicados $happens/2$ e $before/2$ são os únicos abdutíveis e que todas as ações que ocorrem na descrição do domínio são executáveis, em vez de usarmos os metapredicados $abducible$ e $executable$, como no AECP.

Consistência do resíduo negativo Para manter a consistência do resíduo negativo, devemos checá-lo toda vez que o resíduo abdutivo é modificado. Entretanto, num domínio proposicional, precisamos apenas proteger um novo $clipped$ das ações já postuladas no resíduo

H e os $clipped$'s já provados dessa nova ação adicionada. Então, diferentemente do que acontece no AECP, o tratamento de conflitos no ABP é incremental e consome tempo $O(|H|)$. Ademais, quando uma ação é reutilizada para satisfazer alguma precondição, apenas o novo literal $clipped$ adicionado precisa ser protegido. No AECP, que utiliza variáveis, a reutilização de uma ação do resíduo pode causar uma nova unificação que pode fazer com que uma ação já postulada passe a ameaçar a negação de um $clipped$ já provada. Além disso, para tratar um conflito, o AECP usa uma cláusula do tipo $findall(\ldots,(terminates(A,F,T),member(happens(A,T),H)),\ldots)$, que exige que o meta-interpretador encontre todas as ações que terminam um fluente e só depois verifique se tais ações ocorrem no resíduo. Isso requer uma busca exaustiva na base de conhecimentos do PROLOG. Entretanto, partindo da suposição que, em geral, existem muito mais operadores na descrição de um domínio que ocorrências de ações postuladas no resíduo abdutivo, uma tal cláusula faz muito trabalho desnecessário. Como a nova implementação só considera as ações postuladas no resíduo, apenas para essas ações a base de conhecimentos é consultada e, conseqüentemente, a consistência do resíduo é mais eficiente.

A versão sistemática do ABP, o planejador SABP, foi obtida modificando-se o axioma (EC3) de tal modo que não somente as ações que terminam um fluente, mas também aquelas que o iniciam, fossem consideradas ameaças. Isso aumenta o compromisso que o planejador faz com a ação que já produz o fluente, evitando, assim, redundância na satisfação de metas. Tal mudança não compromete, em nada, a proposta original do planejador abdutivo [17, 4].

4.2 Resultados Empíricos

Para avaliar o desempenho dos quatro algoritmos, foram resolvidos problemas em domínios artificiais propostos por *Barret & Weld* [2] e suas extensões propostas por *Kambhampati* [10]. Com isso, visamos garantir que os resultados empíricos obtidos fossem independentes das idiossincrasias de um domínio particular. Originalmente, os domínios D^0S^1, D^1S^1, D^mS^1 e D^mS^2 foram projetados de modo a propiciar conjuntos de submetas *independentes*, *trivialmente-serializáveis*, *laboriosamente-serializáveis* e *não-serializáves*, respectivamente. A notação D^mS^n indica que os operadores no domínio removem m fluentes e que n passos são necessários para a realização de uma submeta. As extensões propostas por *Kambhampati* mantêm essas características, mas tornam os espaços de busca redundantes. Para isso, dois novos fluentes são considerados: os operadores de número par removem he e adicionam hf, enquanto aqueles de número ímpar removem hf e adicionam he. Temos a seguir a especificação para o domínio D^mS^1 e sua extensão D^mS^1R. Os demais domínios são estendidos analogamente.

D^mS^1	$Op(act:A_i, pre:\{I_i\}, add:\{G_i\}, del:\{I_j \mid j<i\})$
D^mS^1R	$Op(act:A_i, pre:\{I_i, hf\}, add:\{G_i, he\}, del:\{I_j \mid j<i\}\cup\{hf\})$, para i ímpar $Op(act:A_i, pre:\{I_i, he\}, add:\{G_i, hf\}, del:\{I_j \mid j<i\}\cup\{he\})$, para i par

Para cada domínio foram definidos 6 operadores e foram gerados 30 problemas distintos, cada um deles consistindo de uma permutação de 6 condições iniciais e de uma permutação de 1 a 6 submetas selecionadas aleatoriamente. Para cada número de metas, foram geradas 5 instâncias e cada ponto nos gráficos representa o tempo médio gasto pelos algoritmos para resolver instâncias de um mesmo tamanho; exceto no último, que mostra o número médio de nós/planos processados durante a busca.

Equivalência das implementações Em todos os testes realizados, além das curvas que relacionam tamanho de problema (número de metas) com tempo médio de CPU, avaliamos

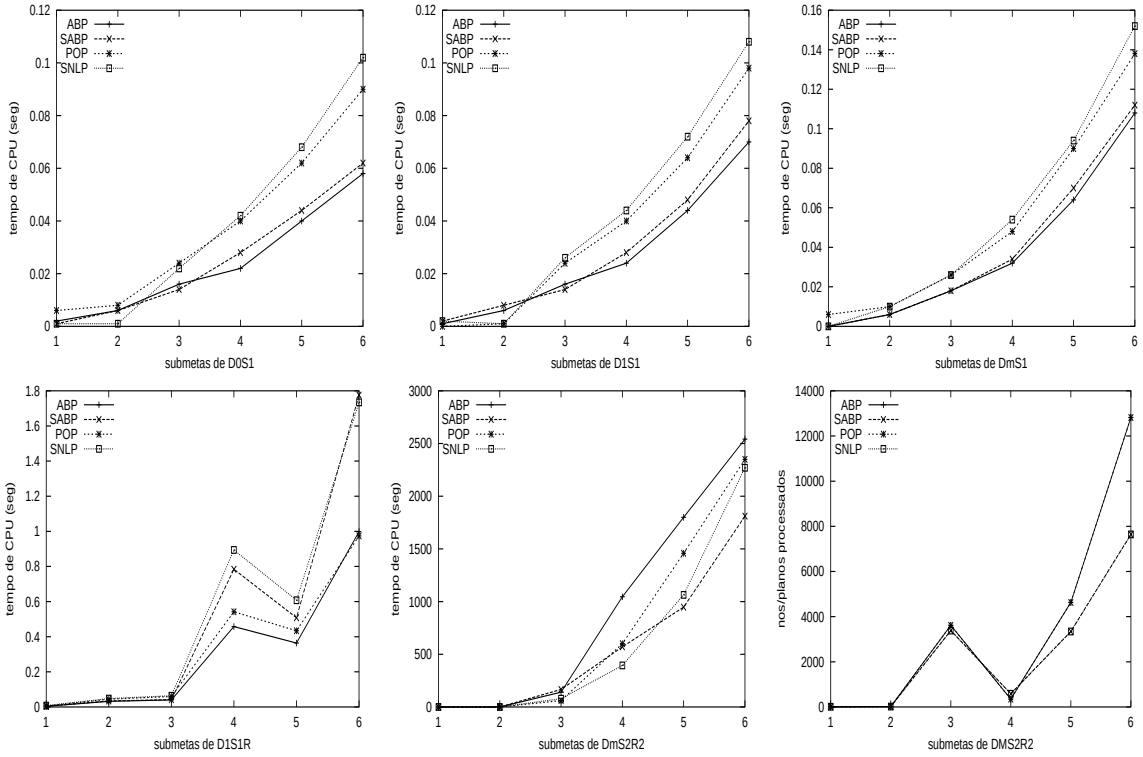

Figura 3: Resultados de alguns dos testes realizados com os domínios artificiais

as curvas que relacionam tamanho de problema com tamanho do espaço de busca (número de nós/planos processados). Observamos que algoritmos equivalentes, ABP/POP e SABP/SNLP, geraram espaços de busca idênticos. Como previsto, esses dados confirmam a equivalência das implementações. De fato, espera-se que métodos de planejamento equivalentes examinem o mesmo número de planos, independentemente de suas implementações. Assim, esse resultado confirma também a correspondência entre os métodos avaliados, verificada por *Shanahan* [17] através de inspeção de código, agora estendida para planejamento sistemático.

Abordagem abdutiva versus algorítmica A figura 3 mostra o tempo médio de CPU gasto para resolver problemas nos domínios D^0S^1, D^1S^1, e D^mS^1. Como podemos observar, nesses domínios, os planejadores sistemáticos não apresentam nenhuma vantagem. Isso era esperado, já que esses domínios não são redundantes. De fato, as versões sistemáticas e não sistemáticas de cada uma das abordagens (abdutiva e algorítmica) apresentam aproximadamente o mesmo desempenho nesses domínios. No entanto, observamos que as duas versões do planejador abdutivo, ABP e SABP apresentaram melhor desempenho que os planejadores clássicos avaliados. Isso pode ser explicado pela representação de ações empregada, que difere de uma abordagem para outra. Ações no cálculo de eventos são representadas por axiomas do tipo $Initiates$ e $Terminates$, o que torna mais eficiente a busca por ações que realizam ou que ameaçam a realização de um determinado fluente. Já na abordagem algorítmica, onde ações são representadas por operadores STRIPS, esta busca deve ser feita em todas as listas de efeitos de todos os operadores. Podemos ainda notar que, para os domínios com metas *independentes*, *trivialmente-serializáveis* e *laboriosamente-serializáveis* (gráficos D^0S^1, D^1S^1 e D^mS^1, respectivamente) todos os planejadores apresentaram complexidade aproximadamente

linear com relação ao número de submetas do problema. Isso também era esperado, uma vez que todos os planejadores implementados são de ordem parcial [2, 10]. A pequena curvatura observada nesses gráficos (herdada nos gráficos para domínios redundantes), advém da complexidade de algumas das operações realizadas em um passo de planejamento (e.g. o fecho transitivo da relação $\prec$, computado a cada nova restrição de ordem temporal adicionada). Em geral, assumimos que o custo de um passo de planejamento é constante.

Sistematicidade A principal motivação por trás da sistematicidade é evitar redundância no espaço de busca explorado pelo planejador [13]. Um corolário importante de sistematicidade é que, no pior caso, um planejador sistemático faz menos busca que um planejador correspondente não sistemático. Sendo $D^m S^2 R^2$ o domínio com maior redundância, podemos considerá-lo como o pior caso, ou seja, problemas nesse domínio são considerados muito difíceis pois envolvem metas laboriosamente serializáveis com muita redundância. Para esse domínio, os planejadores sistemáticos, SABP e SNLP, apresentaram melhor desempenho (gráfico central inferior). Isso pode ser justificado observando-se o número de nós processados (gráfico direita inferior): planejadores sistemáticos processaram um menor número de planos com o crescimento do número de submetas. Esses resultados confirmam o corolário acima, ou seja, planejadores sistemáticos podem ser mais eficientes em domínio que apresentem muita redundância por processarem um menor número de nós. Já para o domínio $D^1 S^1 R$, as diferenças de desempenho não são tão significativas (frações de segundos) e podemos notar uma leve tendência de piora no desempenho dos planejadores sistemáticos. Esse domínio não pode ser considerado como pior caso (envolvendo metas trivialmente serializáveis com muita redundância) e portanto, como esperado, não podemos afirmar nada sobre o desempenho dos planejadores sistemáticos em problemas de complexidade média. Em [10], *Kambhampati* mostra que planejadores que fazem um compromisso entre sistematicidade e redundância possuem, em geral, melhor desempenho em domínios com muita redundância.

5 Conclusões

Neste trabalho foi apresentada uma análise empírica para quatro planejadores que diferem de acordo com o método de planejamento (planejamento parcialmente-ordenado, sistemático e não sistemático) e com a abordagem adotada (lógica ou algorítmica). Para garantir a consistência da análise, todos os planejadores foram implementados de forma a compartilharem, sempre que possível, estruturas de dados e procedimentos. Além disso, aspectos de representação do conhecimento que implicassem em queda de desempenho, devido às características da linguagem de implementação, foram devidamente tratados, sem que a natureza de suas representações fosse alterada (e.g. o cálculo de eventos ou a representação STRIPS). Isso foi confirmado com os dados observados sobre o número de nós processados: planejadores que implementam métodos iguais apresentaram espaços de busca idênticos.

Um dos principais resultados observados é que a construção de um sistema de planejamento baseada em prova de teoremas não introduz aumento de complexidade, desde que se tomem decisões de implementação adequadas. Neste trabalho, tais decisões evitaram o uso de artifícios de programação (e.g. $cuts$ do PROLOG) que afetassem a proposta inicial da abordagem lógica: obter um sistema de planejamento através da especificação do problema de planejamento mais um provador de teoremas de propósito geral, estendido com abdução.

Outro aspecto importante ressaltado com esse trabalho é que, de acordo com resultados

de análises anteriores da comunidade de planejamento em IA, planejadores possuem desempenhos que variam com as caraterísticas do domínio em que se deseja resolver problemas. Desta forma, mostramos que um planejador abdutivo possui o mesmo comportamento que os métodos de planejamento clássico correspondentes, para os diferentes domínios testados. Isso sugere que um planejador abdutivo, assim como qualquer sistema de planejamento, deve ser "calibrado" de acordo com as características do domínio considerado.

Um dos aspectos a serem investigados futuramente é a implementação de uma versão abdutiva para o TWEAK, considerado o mais redundante dos métodos de planejamento por não usar vínculos causais nem adotar estratégias de proteção das submetas realizadas. Conforme *Knoblock & Yang* [11], dependendo da relação existente entre o número de ameaças positivas e negativas, um método redundante como o TWEAK pode ser mais eficiente. Outro aspecto é a implementação de um planejador abdutivo que faça um compromisso entre sistematicidade e redundância, como aquele proposto em [10], através da proteção de múltiplos contribuidores. Com isso esperamos obter melhores resultados em domínios redundantes.

Referências

[1] K. R. Apt, Logic Programming, In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Elsevier Science Publishers B.V. (1990) 493–573.

[2] A. Barret and D. Weld, Partial Order Planning: Evaluating Possible Efficiency Gains, Dept. of Computer Science and Engineering, University of Washigton,Technical Report 92-05-01, July, (1992).

[3] L. N. Barros and P. E. Santos, The Nature of Knowledge in an Abductive Event Calculus Planner, Proc. of 12th International Conference EKAW (2000) 328–343.

[4] L. Chittaro, A. Montanari, Efficient Temporal Reasoning in the Cached Event Calculus, Computacional Intelligence 12 (3) (1996) 359–382.

[5] K. Clark, Negation as Failure, In Herve Gallaire and Jack Minker, editors, Logic and Data Bases, Plenum Press, N.Y. (1978) 293–322.

[6] K. Eshghi, Abductive Planning with Event Calculus, Proc. of 5th International Conference on Logic Programming (1988) 562–579.

[7] R. E. Fikes and N. J. Nilsson, STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, Artificial Intelligence, vol. 2 (3/4) (1971) 189–208.

[8] C. Green, Application of Theorem Proving to Problem Solving, Proc. IJCAI (1969) 219–240.

[9] A. C. Kakas, R. A. Kowalski and F. Toni, Abductive Logic Programming, Journal of Logic and Computation, vol. 2, (1993) 719–770.

[10] S. Kambhampati, Multi-contributor Causal Structures for Planning: A Formalization and Evaluation, Artificial Intelligence, vol. 69, (1992) 235–278.

[11] C. Knoblock and Q. Yang, Evaluating the Tradeoffs in Partial-Order Planning Algorithms, In Proc. Canadian Conference on Artificial Intelligence (1994).

[12] R. A. Kowalski and M. J. Sergot, A Logic-based Calculus of Events, New Generation Computing 4, (1986) 67–95.

[13] D. McAllester and D. Rosenblitt, Systematic Nonlinear Planning, Proc. 9th Nat. Conf. on AI, (1991) 634–639.

[14] J. McCarthy and P. J. Hayes, Some Philosophical Problems from the Standpoint of Artificial Intelligence, In B. Meltzer, D. Michie and M. Swann, ed., Machine Inteligence 4 (1969) 463–502.

[15] C. S. Peirce, Collected Papers of Charles Sanders Peirce, vol. 2, 1931-1958, Harvard University Press.

[16] R. Reiter, On Closed Word Data Bases, In H. Gallaire and J. Minker, Logic and Data Bases, Plenum Press, NY, (1978) 55–76.

[17] M. P. Shanahan, An Abductive Event Calculus Planner, The Journal of Logic Programming, 44 (2000) 207–239.

[18] D. S. Weld, An Introduction to Least Commitment Planning, AI Magazine, 15(4) (1994) 27–61.