Instituto de Matemática e Estatística da Universidade de São Paulo
Departamento de Ciência da Computação
MAC5701: Tópicos em Ciência da Computação

Planejamento & Escalonamento em Inteligência Artificial

por Aldebaran Perseke

Sumário

O presente trabalho tem por objetivo apresentar os frutos dos estudos realizados no 2^o semestre de 2001 nas áreas de planejamento e escalonamento em Inteligência Artificial. Essa monografia apresenta uma introdução às tarefas de planejamento e escalonamento do ponto de vista de Inteligência Artificial, com o objetivo de descrever as classes de problemas de cada uma, bem como apresentar as distinções e similaridades entre essas duas tarefas. Será mostrado também que um problema de planejamento pode ser visto como uma extensão do problema de escalonamento, permitindo que se trate de restrições de recursos e/ou tempo. Finalmente, é apresentada uma proposta de aplicação de um método de planejamento para a resolução de problemas de planajemento que envolvem restrições de recursos. Essa proposta é parte da temática da dissertação de mestrado do autor.

1 Introdução

Planejamento em Inteligência Artificial, ou simplesmente planejamento em `IA`, é a área da Inteligência Artificial que ocupa-se dos problemas de representação de ações, bem como dos algoritmos destinados à tarefa de escolher as ações apropriadas que um agente [1] deve executar para que consiga atingir seus objetivos. Escalonamento em `IA` por sua vez, pode ser descrito como um problema de distribuir tarefas à um número limitado de recursos, e/ou máquinas, de forma que se consiga otimizar uma funções objetivo. Para o problema de escalonamento, as tarefas ou ações que devem ser executadas já devem ser conhecidas de antemão. A primeira vista, as duas tarefas são bastante distintas. Planejamento preocupa-se com **o que** fazer, ou seja, quais devem ser as ações a serem executadas. Escalonamento preocupa-se com **como** fazer, ou seja, em que ordem as ações devem ser executadas para satisfazer um critério de otimização.

Os recentes avanços nas abordagens e nos algoritmos de planejamento em `IA` tem permitido a resolução de problemas extremamente complexos. Com isso, surgiram propostas para tratar problemas que envolvem planejamento e escalonamento em domínios reais, como o controle do telescópio espacial Hubble [12]. Esse trabalho apresentar os dois problemas, bem como uma das abordagens utilizadas para resolver problemas que envolvem planejamento e escalonamento.

2 Planejamento em Inteligência Artificial

2.1 A tarefa de planejamento

Uma característica fundamental de formas de vida inteligentes e sofisticadas, como os seres humanos, é a capacidade de deliberar sobre as ações apropriadas que deve tomar em seu ambiente, para com isso atingir seus objetivos. Para tal, ele raciocina sobre os objetivos que deseja alcançar e as ações que pode tomar para os atingir. Esse raciocínio sobre escolha de ações é chamado de planejamento. Para exemplificar, considere o problema real de planejamento de uma viagem, saindo de São Paulo e chegando à Paris. Esse objetivo consiste de várias etapas, como reservar as passagens aéreas, ir ao aeroporto, trocar reais por francos, fazer a reserva do hotel, encontrar o hotel ao chegar à Paris, etc. Atingir esse objetivo requer a execução de um conjunto complexo de ações na ordem correta, e uma pessoa prudente deveria gastar algum tempo raciocinando sobre essas ações e sua ordem apropriada com antecedência.

Com a finalidade de reproduzir as características desse tipo de comportamento inteligente, a área de planejamento em Inteligência Artificial dedica-se ao estudo de formas de representação dessa classe de problemas, bem como

---

[1] Em Inteligência Artificial, o termo agente é utilizado para designar uma entidade que percebe e age, podendo essa entidade ser puramente lógica, como um *Software*.

a elaboração de algoritmos para solucioná-los. Fundamentalmente, planejamento é uma tarefa de síntese, que envolve a formulação de um curso de ações para atingir alguns objetivos desejados. A complexidade desse problema cresce exponencialmente com a quantidade total de ações que podem ser enumeradas, quando se está selecionando a ação apropriada para a solução de uma etapa do problema. Muitos problemas estão na classe `NP`.

O desafio de construir agentes inteligentes levou os pesquisadores de Inteligência Artificial a investigar algoritmos eficientes para gerar seqüências de ações a fim de solucionar problemas como o descrito. A utilização de algoritmos de planejamento se mostra apropriada à problemas onde um número de açoẽs precisa ser executado em um padrão coerente para atingir um objetivo, ou quando as ações se relacionam de maneira complexa. Os recentes avanços das pesquisas nessa área, aliados a recursos computacionais cada vez mais fartos tem permitido a solução de problemas de domínios cada vez mais complexos, como por exemplo o planejamento de manufatura.

2.2 Descrição do problema

Agentes que planejam possuem uma representação interna do mundo em que estão inseridos. Essa representação é feita através de um modelo de estados. Um estado é simplesmente um conjunto de propriedades, ou predicados que descrevem um instante particular do mundo. Assumindo que existe um número grande, porém finito, de estados em um modelo, chamaremos de $\mathcal{S}$ o conjunto enumerável de todos os estados desse modelo, ou simplesmente espaço de estados. Se o agente encontra-se em um mundo cuja transição entre dois estados é descrita por uma máquina de estados finita determinística, a dinâmica do sistema é definida por uma função de transição de estados f que leva um estado $s_t \in \mathcal{S}$ e uma ação $a_t \in \mathcal{A}(s_t)$ e retorna o próximo estado $f(s_t, a_t) = s_{t+1} \in \mathcal{S}$.

O problema de planejamento é formalmente descrito pela tupla $< \Delta, \Sigma, \Gamma >$, onde:

1. Σ representa a descrição do estado inicial do mundo,
2. Γ representa a descrição do objetivo do agente,
3. Δ representa a descrição das possíveis ações que podem ser escolhidas pelo agente, denominada teoria do domínio.

A tupla $< \Delta, \Sigma, \Gamma >$ é o argumento de entrada para um algoritomo de planejamento, cuja resposta é uma seqüência de ações Π que, quando executadas em qualquer mundo que satisfaça a descrição do estado inicial, irá atingir o objetivo.

Essa formulação para o problema de planejamento é bastante abstrata, pois na verdade ela especifica uma classe de problemas de planejamento parametrizados pelas linguagens utilizadas para representar o mundo, os objetivos e as ações. Outro ponto importante que deve ser notado é que a área de planejamento em IA trata pura e simplesmente de como representar o mundo através de um modelo, e como seqüênciar ações para atingir os objetivo de um agente. Não cabe aqui o estudo do como esse agente percebe o mundo e como ele executa suas ações.

2.3 A linguagem `STRIPS`

A maioria dos planejadores ditos clássicos, descreve estados e operadores em uma linguagem restrita, derivada da lógica de primeira ordem, conhecida como `STRIPS` [2] [3].

Na linguagem `STRIPS`, um estado s_t de um modelo é descrito pela conjunção de n literais l_i

$$s_t : l_1 \wedge l_2 \wedge \cdots \wedge l_n$$

Essa sintaxe assume que todas as variáveis de um estado são universalmente quantificadas.

A descrição de um estado não precisa ser completa, pressupondo que o agente está em um ambiente inacessível, também conhecida como suposição do mundo fechado. Isso significa que $l_i \notin s_t \rightarrow \neg l_i \in s_t$. O objetivo de um agente é uma descrição parcial de um estado do modelo

Operadores são elementos básicos que representam mapeamentos para as reais ações que um agente pode executar. Os operadores disponíveis são agrupados em famílias chamadas de esquemas. Um operador Op de um domínio, identificado pelo nome n_j e por uma lista de m argumentos v_k, é descrito por uma tupla:

$$Op : (n_j(v_1, v_2, \cdots, v_m), p, Ep, En)$$

onde p é a pré-condição do operador, Ep é a lista de efeitos que o operador adiciona ao estado em que é aplicado, e En é a lista de efeitos que o operador remove do estado em que é aplicado. Todas as variáveis de um esquema de operadores são universalmente quantificadas. A pré-condição p e as listas Ep e En seguem as mesmas regras que a descrição do objetivo.

Apenas operadores totalmente instanciados podem ser executados pelo agente. Um operador Op é aplicável em um estado s_t se suas variáveis v_k puderem ser instanciadas de modo que p unifica com uma conjunção de um subconjunto de literais de s_t.

A seguir, algumas simplificações que são adotadas para os modelos de representação do STRIPS:

Tempo de execução atômico: A execução de uma ação é indivisível e ininterrupta, assim não há a necessidade de se considerar os estados

---

[2] A sigla `STRIPS` é a abreviação para "STanford Research Institute Problem Solver", um famoso e influente planejador construido nos anos 70 para controlar um instável robô móvel, afetivamente conhecido como "Shakey"

do mundo enquanto a execução está acontecendo. Portanto, devemos modelar a execução como uma trasnformação atômica de um estado do mundo para um novo estado. Dessa forma, é impossível a execução de duas ou mais ações simultaneamente.

Efeitos determinísticos: O efeito da execução de qualquer ação é uma função determinística da ação e do estado do mundo antes da ação ser executada.

Uniciência: O agente tem o conhecimento completo do estado inicial do mundo e da natureza de suas ações.

Causas únicas de mudança: A única maneira que o mundo pode ser modificado é pelas ações do próprio agente. Não existem outros agentes e o mundo é estático.

Temos que admitir que essas premissas são irreais, porém elas simplificam o problema a tal ponto que podemos descrever alguns algoritmos simples.

2.4 Planejamento Clássico

Em problemas de planejamento clássico, o objetivo é atingir um conjunto de objetivo, usualmente expressos como um conjunto de literais positivos e negativos no cálculo proposicional. O estado inicial também é expresso como um conjunto de literais. As ações possíveis são caracterizadas utilizando operadores `STRIPS`. Um operador `STRIPS` é um modêlo parametrizado de um conjunto de ações possíveis. Ele contém uma pré-condição que precisa estar satisfeita para a aplicação do operador, e um conjunto de efeitos que o operador exerce no mundo. Um número mais recente de sistemas de planejamento tem utilizado uma versão extendida do `STRIPS`, conhecida como `ADL`. O `ADL` permite disjunções nas pré-condições, efeitos condicionais, e quantificação universal limitada nas pré-condições e nos efeitos.

No entanto, existe uma série de limitações nas representações do `STRIPS` e do `ADL`:

- Não existe modelo explícito de representação do tempo
- Não existe provisão para a especificação de requerimento de recursos
- Não existe a possibilidade de representação de incertezas
- Os únicos tipos de objetivos são os objetivo a serem alcançados

Após representado o problema, um número de diferentes técnicas tem sido estudadas para resolver os problemas de planejamento clássico. A abordagem mais óbvia e direta é a busca progressiva no espaço de estados (`FSS`, Forward State Space Search).

Muitos planejadores constróem planos pela busca regressiva a partir dos objetivo. A idéia básica é escolher uma ação que atinja um dos objetivo, e adicionar a ação a um plano parcial. Apesar de todo o esforço investido em planejadores dirigido ao objetivo do planejamento clássico (particularmente o `POCL`, Partial Order Causal Link), esses planejadores não atingiram muito sucesso prático. O sucesso desses planejadores dirigidos ao objetivo depende fortemente de guias heurísticas, como no planejador `FSS`. Sem a ajuda de heurísticas, esses planejadores geralmente são limitados à solução de problemas que envolvem uma ordem de apenas algumas dúzias de ações.

O `Graphplan` emprega uma abordagem diferente na busca por planos. A idéia básica é o emprego de um tipo de análise de alcançabilidade para eliminar muitas das combinações e seqüências de ações que não são compatíveis. Vários dos melhores planejadores apresentados na competição do *AIPS-98* são baseados na técnica do `Graphplan`. A técnica do `Graphplan` tem sido extendida para lidar com operadores com quantificação e efeitos condicionais e outras características do `ADL`. Existem esforços recentes para permitir alguma consideração limitada de tempo e de quantidades métricas no `Graphplan`. Ainda resta ver quão longe essas idéias chegarão e quão bem as técnicas se enquadrarão.

O planejamento também pode ser encarado como satifação, cuja idéia básica é adivinhar o tamanho do plano, e traduzir o problema de planejamento em um conjunto de fórmulas proposicionais, e tentar resolver o problema resultante de satisfação (`SAT`). Os melhores planejadores `SAT` tem desempenho bastante similar aos planejadores `Graphplan`. Assim como esses, os planejadores `SAT` podem ser extendidos para permitir operadores com quantificação e efeitos condicionais. De fato, essas extensões tem impacto apenas no processo de tradução, não no processo de solução. As maiores desvantagens da abordagem do planejador `SAT` é o número muito grande de variáveis e cláusulas para traduzir um problema, e a limitação à descrição apenas de restrições de tempo discreto.

3 Escalonamento

3.1 O que é escalonamento

Escalonamento pode ser definido como um problema de atribuição de um número limitado de recursos à tarefas ao longo do tempo a fim de otimizar uma ou mais funções objetivo. Essa definição sucita três aspectos importantes relativos ao problema de escalonamento: é um processo de tomada de decisão, pois pode envolver escolhas, raciocinar sobre tempo e utilização de recursos é o núcleo dos problemas, e os problemas são quase sempre problemas de otimização. Para que tenhamos um escalonamento válido, as seguintes propriedades devem ser satisfeitas: todas as tarefas são executadas por completo, nenhuma tarefa é executada antes da sua data de disponibilidade, cada máquina executa no máximo uma tarefa por vez, e cada tarefa é atribuida a uma máquina por vez.

O conceito comum de escalonamento em *IA* é de que se trata de um caso especial de planejamento em que as ações já estão escolhidas, reduzindo o problema apenas à determinação de uma ordenação apropriada. Isso infelizamente é uma trivialização do problema de escalonamento.

O exemplo canônico de problema de escalonamento é o problema *Job-Shop*. Nesse problema, há um conjunto de máquinas distintas, cada qual capaz de executar um subconjunto das tarefas existentes, sendo que as tarefas estão descritas por um grafo de precedência que indica em que ordem parcial as tarefas devem ser submetidas às máquinas. O problema *Job-Shop* é um problema que pertence à classe `NP`.

3.2 Descrição do problema

Problemas de escalonamento são formalmente descritos por um conjunto de três campos, $\alpha|\beta|\gamma$. O campo α descreve os recursos, e pode assumir os valores 1, que indica a existência de uma única máquina, P_m indica a existência de m máquinas idênticas, P_∞ que indica a existência de um número ilimitado de máquinas idênticas, ou Q_m que indica a existência de m máquinas com velocidades diferentes. O campo β representa o tempo de preparação de cada máquina, a existência de uma relação de precedência entre as tarefas (representada por um grafo orientado), tempo de comunicação entre as máquinas e restrições quanto a escolha de máquinas para cada tarefa. O campo γ representa o critério de otimização do problema, sendo o mais simples o *makespan*, que é o instante de tempo em que a última tarefa é finalizada.

Uma classe bastante genérica de problemas de escalonamento, conhecida como escalonamento baseado em projeto de restrição de recursos - `RCPS` [3]. Uma instância de problema `RCPS` consiste de:

- Um conjunto finito de tarefas, cada qual com uma certa duração
- Um conjunto finito de recursos, cada qual tendo uma capacidade especificada
- Uma especificação de quanto de cada recurso cada tarefa requer
- Um conjunto de restrições de ordem das tarefas

3.3 Resolvendo o problema

A abordagem mais comum para resolver um problema de escalonamento é representá-lo como um problema de satisfação de restrições (CSP, Constraint Satisfaction Problem) e utilizar uma técnica geral para satisfação de restrições. Duas abordagens tem sido exploradas na modelagem de problemas de escalonamento como problemas de satisfação de restrições. As possibilidades são:

- Atribuir horários para o início de cada tarefa de forma que todas as restrições de tempo e recursos sejam satisfeitas

---

[3] *Resource-Constrained Project Scheduling*

- Impor restrições de ordem entre as tarefas de forma que todas as restrições de tempo e recursos sejam satisfeitas

3.3.1 Escolha do horário de início

Um caminho muito natural de representar problemas RCPS como problemas de satisfação de restrições é:

- Definir uma variável que representa o horário de início de cada tarefa. O domínio da variável consiste de todos os possíveis horários de início em uma intervalo discretizado definindo o horizonte do escalonamento.
- Especificar restrições que forcem a ordenação das tarefas dadas.
- Para cada instante do horário e para cada recurso, especificar as restrições de que o uso total de todas as tarefas ativas no dado instante não excede a capacidade de cada recurso.

As decisões então tomam a forma de atribuição individual de horário de início para cada tarefa. Essa abordagem depende da discretização do tempo, fazendo com que seja necessário definir instantes de tempo atômicos para que o problema possa ser resolvido.

3.3.2 Ordenação das tarefas

A outra maneira usual de representação é baseada na idéia de que duas tarefas que estão ordenadas não irão competir pelo mesmo recurso. Definindo-se a ordenação de variáveis para pares de tarefas, obtem-se a seguinte representação de restrições:

- Uma variável lógica (booleana) para cada par de tarefas, representando que a primeira tarefa precede a segunda.
- Restrições para determinar o horário de início para cada tarefa, baseado na ordenação.
- Restrições forçando que, se cada tarefa é iniciada tão cedo quanto possível, então não haverá conflito no uso dos recursos.

Essa forma de representação permite o uso de propagação de restrições para o rastreamento de possíveis horários de início, desde que as únicas decisões feitas sejam as de operações de ordem. Quando se utiliza uma codificação baseada em ordenação, decisões de ordenação precisam ser feitas apenas até que o sistema possa garantir que não existe conflito no uso de recursos. Entretanto, quando a demanda por recursos se torna mais complexa, a verificação de que não existem conflitos se torna um desafio. Assim, essa abordagem continua tendo pelo menos uma fraqueza significativa. Nos casos em que os requisitos por recursos dependem do **quando** uma tarefa é escalonada, é difícil determinar corretamente se a capacidade de um recurso é excedida conhecendo-se apenas um limite no tempo de execução de cada tarefa.

4 Arquiteturas integradas

No artigo “Bridging the gap between planning and scheduling” [12], é apresentado um estudo detalhado das possíveis maneiras de tratar problemas de planejamento que envolvem escalonamento, e vice-versa. Segundo o artigo, existem fundamentalmente três diferentes maneiras em que planejamento e escalonamento tem sido feitos:

P&E estratificados A abordagem usual tomada pela comunidade de escalonamento é separar planejamento de escalonamento. A escolha das ações é feita antes, então o problema de escalonamento resultante é resolvido.

P&E intercalados A abordagem adotada pelo planejamento `POCL` permite intercalar a escolha de ações com as decisões de ordenação. Tipicamente uma escolha de ação é feita, e os conflitos são resolvidos com outras ações pela imposição de restrições às ações.

P&E homogêneos A abordagem adotada no planejamento `SAT` e no planejamento `ILP` é transformar o problema de planejamento em um problema de escalonamento. No problema de escalonamento resultante não há distinção entre a escolha de ações e as decisões de ordenação, e as decisões são feitas de uma maneira homogênea única.

A terceira opção parece ser uma das mais interessantes, principalmente se forem levados em conta os recentes avanços dos algoritmos de planejamento, conforme descritos no artigo de Daniel Weld [13]. Esse artigo apresenta os algoritmos *Graphplan* e *Satplan*. Esses dois sistemas de planejamento foram integrados em um planejador de extremo sucesso, chamado *Blackbox*. Minha proposta é extender o uso do *Blackbox* para resolver a classe de problemas em que estou interessado. Nos capítulos seguintes, os dois algoritmos são descritos com detalhes.

5 Planejador *Graphplan*

O planejador *Graphplan*, criado por Blum & Furst [1], é visto por alguns pesquisadores como um dos mais interessantes desenvolvimentos recentes na área de planejamento em IA pelos seguintes motivos:

- *Graphplan* é um algoritmo simples e elegante que engendra um planejador extremamente eficiente.
- A representação utilizada pelo *Graphplan* forma a base das mais bem sucedidas codificações de problemas de planejamento em problemas de satisfazibilidade proposicional.

O planejador *Graphplan* alterna entre duas fases: a construção de um grafo de planejamento e a extração de uma solução desse grafo.

5.1 O grafo de planejamento

A construção do grafo de planejamento foi originalmente proposta a domínios cuja descrição é baseada na linguagem *STRIPS* [3], e por hora não me preocuparei com quaisquer extensões.

O grafo de planejamento é um grafo orientado de níveis, ou camadas. Esse grafo tem alternadamente níveis de proposições e níveis de ações. Os níveis pares do grafo representam proposições através de um vértice, sendo o nível zero do grafo formado pelas proposições do estado inicial do problema de planejamento. Os níveis ímpares do grafo representam, também através de vértices, todas as ações aplicáveis, ou seja, aquelas cujas pré-condições são satisfeitas pelas proposições do nível anterior. Um tipo especial de ação chamada $no - op$ é utilizada para a persistência das proposições pelos níveis do grafo, resolvendo de forma algoritmica o problema do quadro [4] [11]. Os vértices contendo as ações em um nível i são conectados por arestas de pré-condição às proposições do nível $i - 1$, e por arestas de adição e remoção às proposições do nível $i + 1$. As arestas sempre terão a direção de um nível menor para um nível maior, indicando a ordem de execução das ações em um plano solução.

Para ilustrar esses e outros conceitos, adotarei o domínio do jantar surpresa proposto por Daniel S. Weld [13] e descrito abaixo:

1. *Estado Inicial*: lixo $\wedge$ maos_limpas $\wedge$ silencio
2. *Objetivo*: jantar $\wedge$ presente $\wedge$ $\neg$ lixo
3. *Teoria do Domínio*:

   $Operador_1$: cozinhar
   Pré-condição: maos_limpas
   Lista adição: { jantar }

   $Operador_2$: embrulhar
   Pré-condição: silencio
   Lista adição: { presente }

   $Operador_3$: carregar_lixo
   Pré-condição: lixo
   Lista remoção: { lixo, maos_limpas }

   $Operador_4$: triturar_lixo
   Pré-condição: lixo
   Lista remoção: { lixo, silencio }

A figura 1 apresenta o grafo de planejamento para esse problema. Note que as arestas não estão orientadas (considerado tácito), assim como não foram representadas como vértices as ações $no - op$. Isso foi feito apenas para simplificar a figura.

5.2 Relações Binárias de Exclusão Mútua

Como veremos adiante, o plano solução para o problema de planejemento será extraído do próprio grafo de planejamento. Entretanto, na forma como é representado não há implicações que um plano solução terá como escolhas todas as ações ou proposições de cada um dos níveis. Qualquer representação consistente de um estado jamais pode conter duas proposição p e $\neg p$, pois p e $\neg p$ são exclusivas mutuamente. O mesmo princípio é apliçável à ações.

---

[4] *Frame Problem*

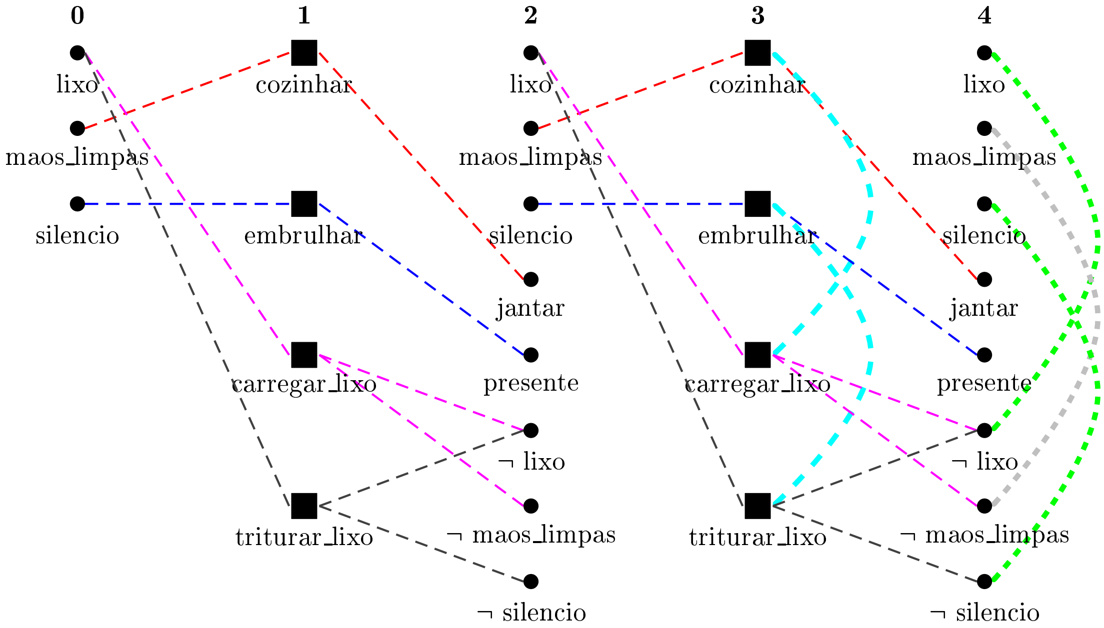

Figura 1: **Grafo de Planejamento** para o domínio do jantar surpresa

Dessa forma, durante a construção do grafo de planejamento, após a adição de cada par de níveis de ações e proposições, o *Graphplan* efetua uma análise de relações de exclusão mútua entre todas as combinações de dois vértices de um mesmo nível do grafo.

Definição 1 *Duas proposições p e q, ambas de um mesmo nível i do grafo de planejamento, são excluvivas mutuamente se não existir um plano válido que as contenha verdadeiras no nível i.*

Da definição 1 temos duas situações para exclusão mútua de proposições: se uma proposição é a negação da outra, ou se todas as maneiras de adicionar uma proposição p são exclusivas mutuamente às maneiras de adicionar a proposição q. Esse último conceito é apresentado na definição 2, lembrando-se que as ações $no-op$ também pode ser utilizadas para adicionar proposições.

Definição 2 *Qualquer proposição é válida em um nível i se e somente se é um efeito de adição ou remoção de alguma ação existente no nível $i-1$.*

Definição 3 *Duas ações A_1 e A_2, ambas de um mesmo nível i do grafo de planejamento, são exclusivas mutuamente se não existir um plano válido que as contenha no nível i.*

Deve-se notar que da definição 3 não há implicação que impeça que ambas as ações pertençam ao plano solução, mas apenas que essas ações não podem pertercer ao mesmo nível em que são exclusivas. As relações de exclusão para ações, de acordo com Daniel S. Weld [13] [5], podem ser de três tipos: efeitos inconsistentes, interferência, e necessidades conflitantes. Exclusão mútua por efeitos inconsistentes acontece quando uma ação adiciona uma proposição que é removida por outra ação. Interferência ocorre quando uma ação remove uma proposição que faz parte da pré-condição de outra ação. Sempre que uma proposição $p_{A,1}$, da pré-condição de uma ação A_1, e uma proposição $p_{A,2}$, da pré-condição de uma ação A_2, são marcadas como exclusivas mutuamente no nível de proposições anterior àquele que contém essas ações, diz-se que essas ações têm necessidades conflitantes.

A figura 1 apresenta as relações de exclusão mútua entre as ações e as proposições dos níveis 3 e 4 através de arcos entre os vértices do mesmo nível. É importante salientar que esses arcos não são arestas do grafo, mas apenas indicações das relações de exclusão.

Durante o processo de busca por um plano solução, os arcos que indicam as proposições e ações exclusivas auxiliam o *Graphplan* a evitar caminhos de busca que apresentam conflitos. Isso portanto reduz o número de nós visitados durante essa busca, garantindo dessa forma a eficiência do algoritmo. Essa é a razão da preocupação com esse tipo de raciocínio sobre interações entre as ações.

5.3 Busca por uma Solução

A partir de um grafo de planejamento cujo último nível contenha todas as proposições do estado objetivo, um plano solução pode ser encontrado através de uma busca de encadeamento reverso por esse grafo. Essa foi a proposta inicial descrita no trabalho de Blum & Furst [1], entretanto, Daniel S. Weld [13] apresentou uma maneira mais eficiente de realizar essa busca. A idéia é resolver um problema dinâmico de satisfação de restrições - *CSP* [6] [2], com variáveis de domínios discretos.

O algoritmo abaixo descreve esse processo de busca:

```
Algoritmo $buscaSolucaoGrafoPlan(\Gamma, G(V,A), E_m)$
Entradas:
    Uma conjunção de proposições do estado objetivo $\Gamma$
    Um grafo de planejamento $G(V,A)$ com $k$ níveis
    Um conjunto de arestas de exclusão mútua entre ações $E_m$
Saída:
    Falha ou um plano solução $\Pi$
```

---

[5] Nesse trabalho Weld estendeu o trabalho inicial de Blum & Furst, e caracterizou melhor as relações de exclusão mútua entre ações

[6] *Constraint Satisfaction Problem*

1. Seja $x = k$

2. Seja Π um plano inicialmente vazio. Π é um conjunto de ações que descreve um plano parcialmente ordenado , cuja ordenação entre ações é dada pelos coeficientes das mesmas

3. Seja P um conjunto de n variáveis p_i, havendo uma correspondência entre p_i e cada proposição Γ_i do objetivo Γ

4. Sejam n conjuntos O_i, cujos elementos são as ações que adicionam a proposição $p_i \in P$ no nível x, ou seja, O_i é o domínio da variável p_i

5. Seja R um conjunto de restrições de ações do tipo:

$$p_i = a_{i,j} \rightarrow p_k \neq a_{k,l}$$

   Tal restrição ocorrerá em R, se e somente se existir uma aresta de exclusão mútua entre as ações a_j e a_l no nível $x - 1$ do grafo de planejamento, aresta pertencente ao conjunto E_m

6. Os conjuntos P, O_i, e R são a entrada para um algoritmo qualquer que resolva o problema *CSP* [2], resolução essa que é um ponto de escolha e portanto ponto para retrocedimento [7] futuro. Se a resolução *CSP* falhar, parar com falha.

7. O resultado dessa resolução é uma atribuição de valores às variáveis do problema *CSP*, descrita pelo conjunto $C_{x/2}$

8. Inserir em Π o conjunto de ações não *no−op* de $C_{x/2}$, com coeficiente $x/2$ (para determinar a ordem em que as ações devem ser executadas)

9. Fazer $x = x - 2$

10. Se $x = 0$, parar com o plano solução Π

11. Faz-se o novo sub-objetivo Γ a conjunção das proposições de pré-condição das ações em $C_{x/2}$

12. Retornar ao passo 3

5.4 O algoritmo *Graphplan*

Conforme dito no início desse capítulo, o planejador *Graphplan* alterna entre a construção do grafo de planejamento e a extração de uma solução desse grafo. Com base nessas duas fases, já descritas nas seções anteriores, o algoritmo *Graphplan* pode ser descrito da seguinte forma:

Algoritmo $Graphplan(\Delta, \Sigma, \Gamma)$

Entradas:

Uma descrição de um domínio Δ
Uma conjunção de proposições do estado inicial Σ
Uma conjunção de proposições do estado objetivo Γ

Saída:

Falha ou um plano solução Π

---

1. Seja $G(V, A)$ um grafo de planejamento inicialmente nulo

2. Seja E_m um conjunto de arestas de exclusão mútua inicialmente vazio

---

[7] *Backtracking*

3. Criar o nível 0 do grafo de planejamento contendo um vértice para cada uma das proposições do estado inicial do problema
4. Iterativamente montar os níveis i de ações e $i+1$ de proposições, até que todas as proposições do estado objetivo estejam presentes no último nível sem que haja mutualidade exclusiva entre elas
5. A partir do grafo de planejamento, realizar uma busca por um plano solução, encerrando se encontrar um plano válido, expandindo o nível seguinte no caso contrário

Dessa maneira, em cada iteração do algoritmo descobre-se um plano válido ou prova-se que não pode existir um plano solução que tenha a quantidade de níveis presente no grafo de planejamento. *Blum & Furst* provaram que o algoritmo é correto e completo.

6 O Planejador *Satplan*

O planejamento clássico é formalizado como um problema de dedução, ou seja, é um processo de encontrar um prova dedutiva de uma expressão que assegure que as condições iniciais seguidas de uma seqüência de ações implique nas condições do objetivo. Para solucionar esse problema vários pesquisadores assumiram que o uso de algoritmos de propósito específico deveria ser necessário a fim de se obter um desempenho satisfatório. Algoritmos como *UCPOP* [6] e mesmo o *Graphplan* podem ser vistos como provadores automáticos de teoremas aplicados ao problema de planejamento.

O problema complementar da dedução é o da satisfazibilidade - `SAT`, que consiste em encontrar um modelo, ou uma interpretação para um conjunto de axiomas. Enquanto a dedução é um problema complexo (comprovadamente `NP` difícil), a área de IA sempre supôs que o problema `SAT` fosse ainda pior. Realmente, para o caso de primeira ordem a prova de teoremas é no mínimo semidecidível, enquanto pode não existir um procedimento completo para encontrar modelos para fórmulas arbitrárias. No caso proposicional, entretanto, parece não haver uma base teórica que afirme que um problema é mais difícil que o outro.

6.1 A Motivação

No início dos anos 90 foram desenvolvidos alguns algoritmos que resolvem problemas `SAT` extremamente grandes e complexos, expressos como um conjunto de cláusulas proposicionais. O algoritmo *GSat* [8] [8] encontra a satisfazibilidade para fórmulas descritas como uma `CNF` [9], ou seja, uma conjunção de cláusulas, que por sua vez são uma disjunções de literais proposicionais positivos ou não. *GSat* pode, por exemplo, resolver empiricamente problemas difíceis de coloração de grafos com um desempenho surpreendente. Motivados pelo sucesso do algoritmo *GSat*, Kautz & Selman formularam o problema do planejamento em IA como satisfazibilidade proposicional [5], engendrando então o planejador *Satplan*.

---

[8] *Greedy local search procedure*

[9] *Conjunctive Normal Form*

A mudança na abordagem beneficia-se também do fato de que diversos pesquisadores de outras áreas contribuem para o aprimoramento dos algoritmos de satisfazibilidade, que podem assim ser incorporados ao planejadores [10]. Só para citar um outro exemplo, o algoritmo *WalkSat* [7], que é um aprimoramento do algoritmo *GSat* , é ainda mais eficiente que este último, resolvendo fórmulas com vários milhares de variáveis proposicionais.

Além disso, a abordagem do planejamento como um problema `SAT` apresenta as seguintes vantagens sobre as abordagens clássicas: é fácil especificar condições em qualquer estado intermediário do mundo, e pode-se adicionar condições que incluem eventos que estão além do alcance do agente.

6.2 O Algoritmo

O planejador *Satplan*, que possui uma abordagem totalmente diferente dos planejadores clássicos, ainda utiliza a linguagem `STRIPS` para descrever o domínio e o problema a ser resolvido. Como o seu processo para encontrar um plano solução é baseado em sentenças lógicas que quer satisfazer, o primeiro passo a ser tomado é o de transformar, ou compilar a representação `STRIPS` em uma representação lógica. Durante essa compilação do problema original, *Satplan* cria uma fórmula `CNF` com a descrição do estado inicial, a descrição das possíveis ações que podem ser escolhidas em cada instante do plano, e a descrição do estado objetivo. Como o comprimento do plano solução não é conhecido de antemão, *Satplan* constrói uma fórmula que representa um possível plano com uma comprimento arbitrário, e tenta satisfazê-la um número finito de vezes. Caso a tentativa de satisfazer a fórmula não seja bem sucedida, o comprimento do possível plano é então iterativamente incrementado, até que uma fórmula satisfazível seja encontrada, ou que se alcance o valor de um comprimento de corte. Se uma fórmula satisfazível foi encontrada, a regressão da representação lógica para a representação `STRIPS` deve ser efetuada, a fim de que o planejador forneça como resposta um plano descrito como uma seqüência de ações. A figura 2 apresenta em blocos a arquitetura do planejador *Satplan*.

6.3 Compilação do um Problema em uma Fórmula Proposicional

Na abordagem do planejamento como satisfazibilidade, o problema de planejamento não é um teorema à ser provado, mas simplesmente um conjunto de axiomas com a propriedade de que qualquer modelo dos axiomas corresponde um plano válido. Dessa forma, conforme já foi descrito anteriormente, para que um um problema de planejamento seja resolvido como um problema `SAT`, é necessário codificar o problema como um conjunto de axiomas. Para tal, utiliza-se um modelo padrão de fluentes, onde o tempo assume valores inteiros positivos.

---

[10] A biblioteca de algoritmos de satisfazibilidade, `Satlib` - http://www.satlib.org, é uma ótima fonte dessas contribuições

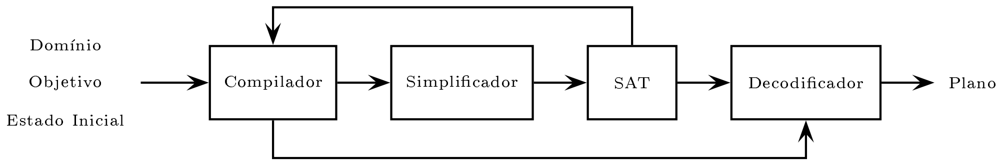

Figura 2: Arquitetura do planejador *Satplan*

A melhor formalização conhecida para o planejamento é o cálculo de situações [9]. A disvantagem do cálculo de situações para os propósitos práticos de planejamento é ser inerentemente de primeira ordem: um infinito número de termos de estados podem ser construidos pelo repetido uso da função resultado [11]

Há várias maneira de se compilar o problema de planejamento em uma `CNF`, e o *Satplan* originalmente propôs a codificaç ao que mais tarde foi denominada codificação linear [4]. A codificação linear é muito semelhante a versão proposicional do cálculo de situações, mas com axiomas adicionais para evitar "modelos anômalos", que não ocorrem no planejamento dedutivo mas são problemáticos no planejamento como satisfazibilidade. A codificação linear utiliza os seguintes tipos de cláusulas:

1. O estado inicial é completamente especificado, ou seja, não há a conhecida suposição do mundo fechado.

   lixo(1) ∧ maos_limpas(1) ∧ silencio(1) ∧ ¬ jantar(1) ∧ ¬ presente(1)

2. O estado objetivo pode ser totalmente ou parcialmente especificado.

   jantar(2) ∧ presente(2) ¬ ∧ lixo(2)

3. Se uma ação acontece em um instante de tempo i, suas pré-condições devem ser válidas no instante i, os fluentes de sua lista de adição devem ser verdadeiros no instante $i + 1$, e os fluentes de sua lista de remoção devem ser falsos, ou negados no instante $i + 1$. Note-se que os efeitos de remoção não são descritos por proposições negadas. Isso porquê, devido ao problema do quadro, não há persistência dos fluentes existentes em um instante i para o instante $i + 1$, a menos que isso seja explicitamente representado. Dessa maneira, segundo a descrição acima as proposições que são removidas por uma ação no instante i simplesmente deixam de ser persistidas no instante $i + 1$.

---

[11] No cálculo de situações, a função resultado, $results(a, s)$, representa a nova situação obtida a partir de uma situação s pela aplicação de uma ação a.

Para remover a possibilidade de uma ação ser executada quando suas pré-condições são falsas, faz se com que a ação implique em suas pré-condições bem como seus efeitos.

$\forall\ I \cdot \text{cozinhar}(I) \rightarrow (\ \text{maos\_limpas}(I) \wedge \text{lantar}(I+1)\ )$
$\forall\ I \cdot \text{embrulhar}(I) \rightarrow (\ \text{silencio}(I) \wedge \text{presente}(I+1)\ )$
$\forall\ I \cdot \text{carregar\_lixo}(I) \rightarrow \text{lixo}(I)$
$\forall\ I \cdot \text{triturar\_lixo}(I) \rightarrow \text{lixo}(I)$

4. O problema do quadro [12] [11] deve ser explicitamente resolvido nessa abordagem. Isso significa que, se uma ação não adiciona ou remove um fluente qualquer, esse fluente deve ser persistido entre os instantes i e $i+1$ caso essa ação acontece no instante i.

   $\forall\ I \cdot \text{cozinhar}(I) \wedge \text{lixo}(I) \rightarrow \text{lixo}(I+1)$
   $\forall\ I \cdot \text{cozinhar}(I) \wedge \neg\ \text{lixo}(I) \rightarrow \neg\ \text{lixo}(I+1)$
   $\forall\ I \cdot \text{cozinhar}(I) \wedge \text{maos\_limpas}(I) \rightarrow \text{maos\_limpas}(I+1)$
   $\forall\ I \cdot \text{cozinhar}(I) \wedge \neg\ \text{maos\_limpas}(I) \rightarrow \neg\ \text{maos\_limpas}(I+1)$
   $\forall\ I \cdot \text{cozinhar}(I) \wedge \text{silencio}(I) \rightarrow \text{silencio}(I+1)$
   $\forall\ I \cdot \text{cozinhar}(I) \wedge \neg\ \text{silencio}(I) \rightarrow \neg\ \text{silencio}(I+1)$
   $\forall\ I \cdot \text{cozinhar}(I) \wedge \text{presente}(I) \rightarrow \text{presente}(I+1)$
   $\forall\ I \cdot \text{cozinhar}(I) \wedge \neg\ \text{presente}(I) \rightarrow \neg\ \text{presente}(I+1)$

5. Uma única ação deve ser executada a cada instante.

   $\forall\ I \cdot \text{cozinhar}(I) \rightarrow \neg\ \text{embrulhar}(I) \wedge \neg\ \text{carregar\_lixo}(I) \wedge \neg\ \text{triturar\_lixo}(I)$
   $\forall\ I \cdot \text{embrulhar}(I) \rightarrow \neg\ \text{cozinhar}(I) \wedge \neg\ \text{carregar\_lixo}(I) \wedge \neg\ \text{triturar\_lixo}(I)$
   $\forall\ I \cdot \text{carregar\_lixo}(I) \rightarrow \neg\ \text{cozinhar}(I) \wedge \neg\ \text{embrulhar}(I) \wedge \neg\ \text{triturar\_lixo}(I)$
   $\forall\ I \cdot \text{triturar\_lixo}(I) \rightarrow \neg\ \text{cozinhar}(I) \wedge \neg\ \text{embrulhar}(I) \wedge \neg\ \text{carregar\_lixo}(I)$
   $\forall\ I \cdot \text{cozinhar}(I) \vee \text{embrulhar}(I) \vee \text{carregar\_lixo}(I) \vee \text{triturar\_lixo}(I)$

Além desses axiomas, o plano deve ser descrito por uma narrativa, como a apresentada abaixo:

$\text{lixo}(1) \wedge \text{maos\_limpas}(1) \wedge \text{silencio}(1) \wedge \neg\ \text{jantar}(1) \wedge \neg\ \text{presente}(1) \wedge (\ \text{cozinhar}(1) \vee \text{embrulhar}(1) \vee \text{carregar\_lixo}(1) \vee \text{triturar\_lixo}(1)\ ) \rightarrow \text{jantar}(2) \wedge \text{presente}(2) \neg \wedge \text{lixo}(2)$

7 Conclusão

Há muitas pessoas de institutos importantes no mundo inteiro interessados nos problemas e algoritmos aqui descritos. A própria NASA, Agência Espacial Americana, tem pesquisadores que dedicam-se exclusivamente à estudos sobre integração de planejamento e escalonamento. Os problemas são complexos e vital importância para muitos sistemas autônomos, como por exemplo o telescópio Hubble ou o robô Soujornei. Infelizmente, não é muito fácil encontrar artigos sobre o assunto. Ainda há muito que ser feito nesse campo, e espero poder dar minha contribuição com a proposta de minha dissertação.

---

[12] *Frame Problem*

Referências

[1] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2):281–300, 1997.

[2] Rina Dechter. Constraint satisfaction.

[3] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application for theorem proving to problem solving. In *Advance Papers of the Second International Joint Conference on Artificial Intelligence*, pages 608–620, Edinburgh, Scotland, 1971.

[4] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 374–384. Morgan Kaufmann, San Francisco, California, 1996.

[5] Henry Kautz and Bart Selman. Planning as satisfiability. In J. LLoyd, editor, *Proceedings of the Tenth European Conference on Artificial Intelligence*, pages 359–379, 1992.

[6] J. Scott Penberthy and Daniel S. Weld. "UCPOP: A sound, complete, partial order planner for ADL". In Bernhard Nebel, Charles Rich, and William Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 103–114. Morgan Kaufmann, San Mateo, California, 1992.

[7] Bart Selman, Henry Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *AAAI-92: Proceedings 10th National Conference on AI.* American Mathematical Society, 1995.

[8] Bart Selman, Hector J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, 1992. AAAI Press.

[9] Murray Shanahan. *Logical Foundations*, chapter 2, pages 27–56. In [10], 1997.

[10] Murray Shanahan. *Solving the Frame Problem.* The MIT Press, 1997.

[11] Murray Shanahan. *What Is the Frame Problem*, chapter 1, pages 1–26. In [10], 1997.

[12] D. Smith, J. Frank, and A. Jónsson. Bridging the gap between planning and scheduling, 2000.

[13] Daniel S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.