ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Departamento de Engenharia de Computação e Sistemas Digitais

PCS2428 - Inteligência Artificial

Seminário

Planejamento Clássico II

Grupo 3

5175119	Douglas Seiichi Onodera
5174907	Milton Eiji Kato
5212256	Pedro Manoel Fabiano Alves Evangelista

Sumário

1. Introdução

No mundo real, o planejamento é feito para gerenciar logística e agendar tarefas, como por exemplo, determinar os passos de uma linha de produção. Esses problemas, por serem mais complexos, exigem uma extensão do modelo de planejamento simples, tanto na representação como na maneira como eles interagem com o ambiente.

Esse seminário se foca nos aspectos restritivos dos modelos mais complexos dos planejadores, que são o tempo e os recursos, e como fazer o agendamento das tarefas subdividindo-as em tarefas de nível mais alto, de maneira hierárquica.

2. Linguagem de descrição STRIP

Uma linguagem utilizada para descrição de planejadores é a STRIPS (**St**anford **R**esearch **I**nstitute **P**roblem **S**olver), brevemente descrita abaixo.

Uma STRIPS é composta pelos seguintes itens:

- Um estado inicial;
- Estados de objetivo;
- Um conjunto de ações.

2.1. Estados

São representados como um conjunto de literais. Literais representam as diversas entidades envolvidas na descrição do problema e que juntos compõe um estado do problema. Podem ser proposicionais, como `Pobre ^ Desconhecido`, ou de primeira ordem, como `Em(P1, São Paulo) ^ Em(P2, Rio)`. Literais de primeira ordem não permitem funções ou variáveis, como em `Em(x,y)` ou `Em(Pai(Fred), Sidney)`.

2.2. Ações

Ações são representadas por um conjunto de literais alterados, a pré-condição e o efeito causado. Por exemplo, um avião que voa de um aeroporto ***de*** para um aeroporto ***para*** tem a seguinte ação:

```
Ação(Voar(p, de, para),
      PRECOND:Em(p, de) ^ Avião(p) ^ Aeroporto(de) ^Aeroporto(para),
      EFFECT: ¬Em(p, de) ^Em(p, para))
```

3. Tempo, agendamento e recursos

A representação STRIPS fala sobre *como* uma ação ocorre, mas não quando nem quanto tempo a ação demora a ser executada. Não há a variável tempo na representação, sendo que o máximo que é representado é a ordem em que as ações são executadas.

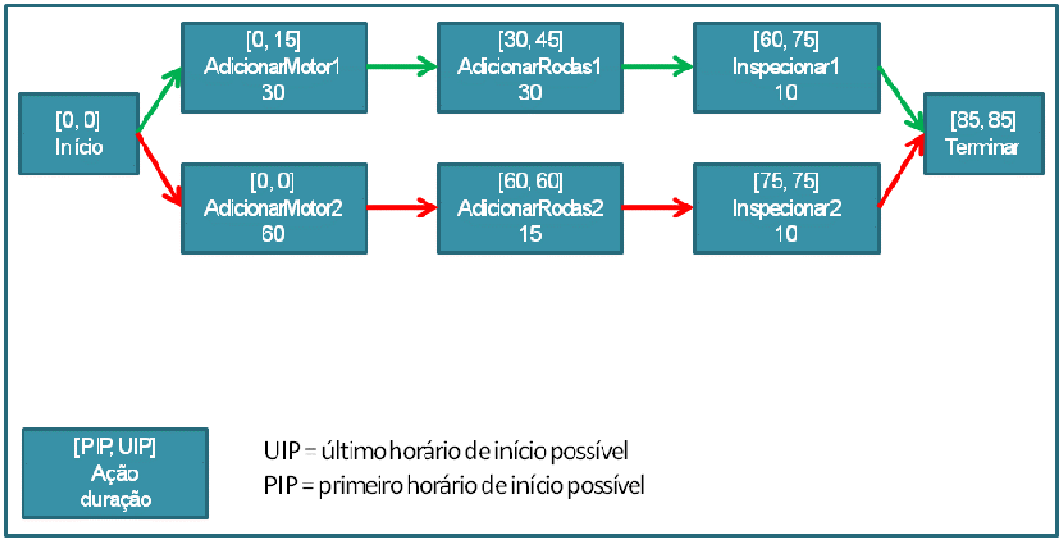
Essa restrição não permite que problemas mais complexos, em que o tempo é um fator crítico, sejam representados. Esse é o caso de aplicações do tipo *Agendamento de Loja de Trabalhos* (job shop scheduling), que consiste numa série de tarefas que demandam um tempo e dependam de um recurso para serem executadas.

Imagine uma montadora de carros, que monta dois carros diferentes, C1 e C2. Cada carro é composto de um chassi, um motor e uma roda. O motor deve ser colocado primeiro que as rodas e o carro é inspecionado somente quando o motor e as rodas estão instaladas.

Queremos que a montagem dos dois carros gaste o menor tempo possível. Abaixo temos uma representação do problema em STRIPS, expandido para incluir o tempo de duração de cada ação. Sem a duração, teríamos somente um problema de planejamento.

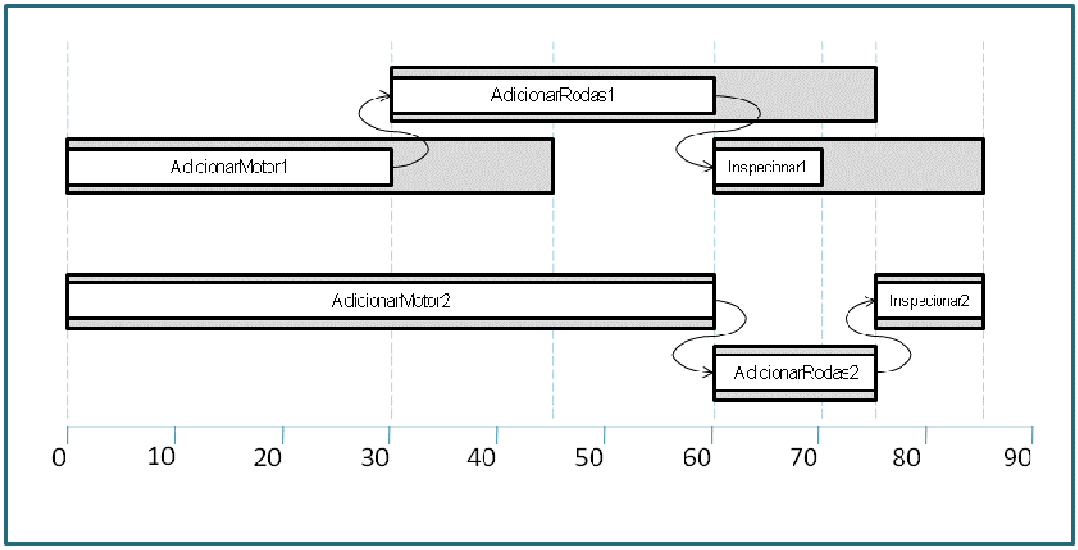
```
Iniciar(Chassi(C1) ^ Chassi(C2)
      ^ Motor(E1, C1, 30) ^ Motor(E2, C2, 60)
      ^ Rodas(W1, C1, 30) ^ Rodas(W2, C2, 15))
Objetivo(Terminado(C1) ^ Terminado(C2))
Ação(AdicionarMotor(e, c, m),
      PRECOND: Motor(e, c, d) ^ Chassi(c) ^ ¬MotorColocado(c),
      EFFECT: MotorColocado(c) ^ Duracao(d))
Ação(AdicionarRodas(w, c),
      PRECOND: Rodas(w, c, d) ^ Chassi(c),
      EFFECT: RodasMontadas(c) ^ Duracao(d))
Ação(Inspecionar(c),
      PRECOND: MotorColocado(c) ^ RodasMontadas(c) ^ Chassi(c),
      EFFECT: Terminado(c) ^ Duracao(10))
```

Dado o ordenamento parcial das tarefas com duração, é possível utilizar o método do caminho crítico para resolver o problema. Um caminho é uma sequência de tarefas entre o Início e Fim do grafo de planejamento parcial. Abaixo, temos o problema escalonado.



3.1. Caminho crítico

O caminho crítico é aquele que não apresenta folga no início do tempo de execução das tarefas que o compõe, ou seja, é aquele que vai determinar quanto tempo demora a execução das tarefas da aplicação. Podemos ver no gráfico abaixo que o caminho crítico para a montagem de carros é a montagem do carro C2.



3.2. Folga e agendamento

A folga representa quanto tempo uma tarefa pode atrasar o seu início sem modificar o tempo total de execução. Para calcular a folga, utilizamos o seguinte algoritmo:

- PIP(Início) = 0
- PIP(B) = $\max_{A,B}$ PIP(A) + Duração(A)
- UIP(Terminar) = PIP(Terminar)
- UIP (A) = $\min_{A,B}$ UIP(B) - Duração(A)

PIP é o primeiro tempo de início possível e UIP é o último tempo de início possível. É fácil determinar que a folga é calculada por UIP – PIP. Basicamente, o que esse algoritmo faz é percorrer os caminhos do início para o fim, fazendo os PIP's das tarefas iguais à soma do maior PIP das tarefas predecessoras mais o tempo de duração da tarefa, e depois percorrer os caminhos do fim para o início, fazendo os UIP's iguais à subtração do menor UIP das tarefas sucessoras com o tempo de duração da tarefa.

3.2.1. Exemplo do cálculo do PIP

- Início

  PIP(Iniciar) = 0

- Passo 1

  PIP(AM1) = max(PIP(Iniciar) + Duração(Iniciar)) = 0

  PIP(AM2) = max(PIP(Iniciar) + Duração(Iniciar)) = 0

- Passo 2

  PIP(AR1) = max(PIP(AM1) + Duração(AM1)) = 0 + 30 = 30

  PIP(AR2) = max(PIP(AM2) + Duração(AM2)) = 0 + 60 = 60

- Passo 3

  PIP(INS1) = max(PIP(AR1) + Duração(AR1)) = 30 + 30 = 60

  PIP(INS2) = max(PIP(AR2) + Duração(AR2)) = 60 + 15 = 75

- Passo 4

  PIP(Terminar) = max(PIP(INS1) + Duração(INS1), PIP(INS2) + Duração(INS2)) = max(60 + 10, 75 + 10) = 85

3.2.2. Exemplo do cálculo do UIP

- Início

  UIP(Terminar) = PIP(Terminar) = 85

- Passo 1

  UIP(INS1) = min(UIP(Terminar) - Duração(INS1)) = 85 - 10 = 75

  UIP(INS2) = min(UIP(Terminar) - Duração(INS2)) = 85 - 10 = 75

- Passo 2

  UIP(AR1) = min(UIP(INS1) - Duração(AR1)) = 75 - 30 = 45

  UIP(AR2) = min(UIP(INS1) - Duração(AR2)) = 75 - 15 = 60

- Passo 3

  UIP(AM1) = min(UIP(AR1) - Duração(AM1)) = 45 – 30 = 15

  UIP(AM2) = min(UIP(AR2) - Duração(AM2)) = 60 – 60 = 0

- Passo 4

  UIP(Iniciar) = min(UIP(AM1) - Duração(Iniciar), UIP(AM2) + Duração(Iniciar)) = min (15, 0) = 0

O conjunto do grafo de planejamento mais os tempos de início possíveis formam o agendamento das tarefas.

3.3. Restrição de recursos

O uso da duração das tarefas não é o suficiente para descrever alguns tipos de problemas que apresentam restrições de recursos. Um recurso é um pré-requisito para uma tarefa, mas ele não é gasto quando a ação é executada. Ao término de uma ação, um recurso volta a ficar disponível para ser usado por outras ações. Chamamos isso de *recurso utilizável*.

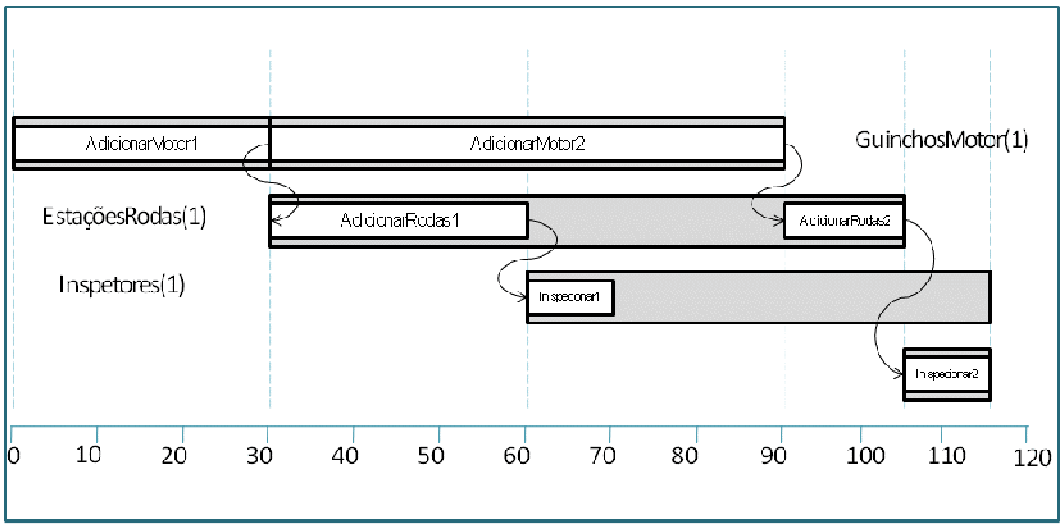
Recursos geralmente não são distinguíveis entre um e outro, por exemplo, se houver dois objetos de um recurso, A e B, não faz diferença se a ação utiliza o recurso A ou o recurso B. Nesse caso, agrupamos esse recurso em um único item. O agrupamento reduz a complexidade do problema, pois a ordem com que os recursos são utilizados não importa.

Voltando ao problema da montagem de carro, suponhamos que para instalar o motor, é necessário utilizar um guincho para levantá-lo, que para colocar as rodas é necessário ter uma estação de instalação de rodas e que para inspecionar o carro é necessário haver um inspetor livre. Precisamos representar isso na descrição do problema. Os recursos são representados como literais de primeira ordem com números que representam a quantidade.

Abaixo, temos a descrição do problema do carro com restrições de recursos:

```
Iniciar(Chassi(C1) ^ Chassi(C2)
      ^ Motor(E1, C1, 30) ^ Motor(E2, C2, 60)
      ^ Rodas(W1, C1, 30) ^ Rodas(W2, C2, 15))
      ^ GuinchosMotor(1) ^ EstaçõesRodas(1) ^ Inspetores(2))
Objetivo(Terminado(C1) ^ Terminado(C2))
Ação(AdicionarMotor(e, c, m),
      PRECOND: Motor(e, c, d) ^ Chassi(c) ^ ¬MotorColocado(c),
      EFFECT: MotorColocado(c) ^ Duracao(d))
      RESOURCES: GuinchosMotor(1))
Ação(AdicionarRodas(w, c),
      PRECOND: Rodas(w, c, d) ^ Chassi(c),
      EFFECT: RodasMontadas(c) ^ Duracao(d))
      RESOURCES: EstaçõesRodas(1))
Ação(Inspecionar(c),
      PRECOND: MotorColocado(c) ^ RodasMontadas(c) ^ Chassi(c),
      EFFECT: Terminado(c) ^ Duracao(10),
      RESOURCES: Inspetores(1))
```

Temos agora os recursos GuinchosMotor(1), EstaçõesRodas(1) e Inspetores(2). Uma possível resolução para o problema é apresentada no gráfico abaixo:



Um problema desse tipo é NP-difícil, o que significa que ele pode ser resolvido por um algoritmo de complexidade não polinomial. O problema do caixeiro viajante é um problema NP-difícil. Um algoritmo de resolução deve levar em consideração o compromisso entre tempo de execução e a solução ótima. Nem sempre a solução ótima pode ser encontrada em um tempo razoável

Muitas abordagens foram feitas para tentar encontrar a melhor solução, sendo uma delas o uso da *heurística da menor folga.*O agendamento das ações é feito de maneira gulosa. Em cada iteração, são consideradas as ações que tiveram todas as suas ações predecessoras agendadas e agenda aquela que tem a menor folga para os inícios mais cedo possíveis. Os parâmetros PIP e UIP de cada ação afetada são então recalculados e a iteração é repetida. A heurística é baseada no mesmo princípio da heurística da variável mais restrita nos problemas de satisfação de restrições.

A abordagem abordada pelo livro é a de planejar primeiro e agendar depois, sendo essa abordagem comum em problemas de logística e manufatura do mundo real, onde a fase de planejamento geralmente é feita por humanos especialistas no tipo de problema.

4. Rede Hierárquica de Tarefas

Uma das idéias mais comuns para se lidar com complexidade é a ***Decomposição Hierárquica.*** Em cada nível da hierarquia, uma tarefa computacional é reduzida a um menor número de atividades no próximo nível inferior. Isso acarreta em menor custo computacional para encontrar uma maneira correta de arranjar estas atividades para solucionar o problema. Métodos não hierárquicos, por sua vez, reduzem uma tarefa para um maior número de ações individuais. Para problemas de larga escala, isto é completamente não prático.

No planejamento baseado na Rede Hierárquica de Tarefas (HTN, de *Hierarquical Task Network*), o plano inicial que descreve o problema é visto como uma ação de altíssimo nível, como por exemplo, "Criar uma casa". Os planos são refinados aplicando-se ***decomposições de ação***. Cada decomposição reduz uma ação de alto nível em um conjunto de ações de menor nível. Este processo continua até que restem apenas ***ações primitivas*** no plano, onde ações primitivas são

aquelas em o executor pode executar automaticamente. No nosso exemplo, a ação "instalar jardinagem" pode ser considerada primitiva, pois isto envolve simplesmente chamar o empreiteiro de jardinagem. Para este empreiteiro, ações como "plantar rododendros aqui" podem ser consideradas primitivas.

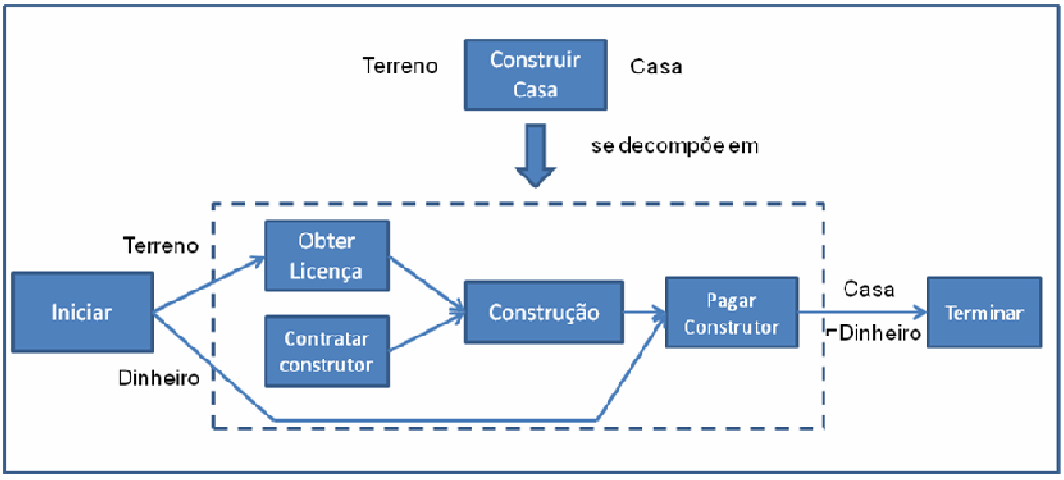
Vale lembrar que as decomposições de ação requerem conhecimento sobre *como* implementar as ações. Por exemplo, "Construir uma casa" pode ser reduzida em "Obter licença", "Contratar uma empreiteira", "Executar a construção" e "Pagar a empreiteira".

4.1. Representando decomposições de ação

A biblioteca de ações é onde se armazena as descrições dos métodos de decomposições de ações, da onde são extraídas e instanciadas as decomposições necessárias no plano a ser construído. Cada método é uma expressão na forma *Decompose(a,d)*, o que significa que a ação *a* pode ser decomposta no plano *d*, que é representado como um plano parcialmente ordenado.

O exemplo da construção de uma casa será usada para ilustrar o conceito da decomposição de ação. A ação *Iniciar* fornece todas aquelas pré-condições das ações no plano que não são fornecidas por outras ações. Chamamos estas condições de ***pré-condições externas***. No exemplo, as pré-condições externas da decomposição são *Terreno* e *Dinheiro*. Analogamente, os ***efeitos externos***, que são as pré-condições da ação Terminar, são todos aqueles efeitos de ações no plano que não são negadas por nenhuma outra ação; No nosso exemplo, os efeitos externos de "Construir Casa" são *Casa* e ⌐*Dinheiro*. Alguns planejadores HTN fazem a distinção entre ***efeitos primários***, como a *Casa*, e ***efeitos secundários***, como o ⌐*Dinheiro*. Apenas efeitos primários devem ser usados para alcançar metas, considerando que

levar em conta ambos os tipos de efeitos pode acarretar em conflitos com outras ações.



Uma biblioteca de ações poderia conter várias decomposições para uma dada ação de alto nível; por exemplo, pode existir outra decomposição para "Construir Casa" que descreve o processo como um agente que constrói a casa a partir de pedras e grama utilizando suas próprias mãos. Cada decomposição deveria ser um plano correto, mas ela poderia ter pré-condições e efeitos além daqueles descritos na sua ação de nível mais alto. Por exemplo, a decomposição de "Construir Casa" na figura acima requer *Dinheiro* e *Terreno* e tem o efeito ¬*Dinheiro*. Por outro lado, na opção de construir tudo sozinho a partir do nada, não requer dinheiro, mas requer um fornecimento de *Pedras* e *Gramado*, que pode resultar em um *BadBack*.

Dado que uma ação de alto nível, como "Construir Casa", pode ter várias decomposições possíveis, é inevitável que a sua descrição STRIPS irá ocultar algumas das pré-condições e efeitos daquelas decomposições. As pré-condições da ação de alto nível devem ser a *interseção* das pré-condições externas de suas decomposições, e os seus efeitos devem ser a interseção dos efeitos de suas decomposições.

Duas outras formas de ocultação de informação podem ser notadas. Primeiro, uma descrição de alto nível ignora completamente todos os ***efeitos internos*** das decomposições. Por exemplo, a decomposição da ação "Construir Casa" possui os efeitos internos temporários Licença e Contrato. Segundo, uma descrição de alto nível não especifica os intervalos "dentro" das atividades durante quais das pré-condições e efeitos de alto nível devem aguardar. No exemplo, a pré-condição *Terreno* precisa ser verdadeira apenas até *Obter Licença* seja executada, e o efeito *Casa* é verdadeiro apenas após a execução de *Pagar Construtor*.

Ocultação de informações deste tipo é essencial se p planejamento hierárquico será usado para reduzir complexidade; precisamos poder raciocinar sobre ações de alto nível sem se preocupar com detalhes de implementação. Há porém, um preço a pagar. Por exemplo, conflitos podem existir entre condições internas de uma ação de alto nível e ações internas de outra, mas não há como detectá-los pelas descrições de alto nível. Este problema leva a implicações importantes nos algoritmos de planejamento HTN. Resumindo, considerando que ações primitivas podem ser tratadas como eventos pontuais no algoritmo de planejamento, ações de alto nível possuem extensões temporárias dentro do qual qualquer evento pode estar ocorrendo.

5. Modificando o Planejamento para decomposições

Apesar de toda ação STRIPS poder ser escrita através de decomposição, o planejamento HTN puro muitas vezes pode se tornar um tanto não natural. Por essa razão, é freqüente utilizar uma forma híbrida de planejamento, utilizando em conjunto a decomposição com o planejamento de ordem parcial.

Para isso, é necessário modificar a função sucessora POP de modo a permitir a aplicação das decomposições no plano parcial atual. Dessa forma, os novos planos sucessores são formados selecionando uma ação não-primitiva no plano e aplicando qualquer decomposição na biblioteca de ações que seja compatível.

Essa rotina pode ser descrita em três passos:

1. Remover a ação de alto nível do plano e, para cada passo da decomposição escolhida, instanciar uma nova ação ou utilizar uma já existente, o que é chamado de sub-task sharing.

2. Conectar as restrições de ordenação. Apesar de aparentemente todas as ações da decomposição devessem ocorrer após as ações anteriores da ação de alto-nível e antes das ações posteriores, isso não é verdade. Algumas ações da decomposição podem ocorrer antes das ações anteriores, e restringir elas a virem depois poderia eliminar possíveis soluções. A melhor maneira de se resolver isso é guardar para cada restrição, as suas razões, permitindo assim que as novas restrições de ordenação sejam as mais leves possíveis.

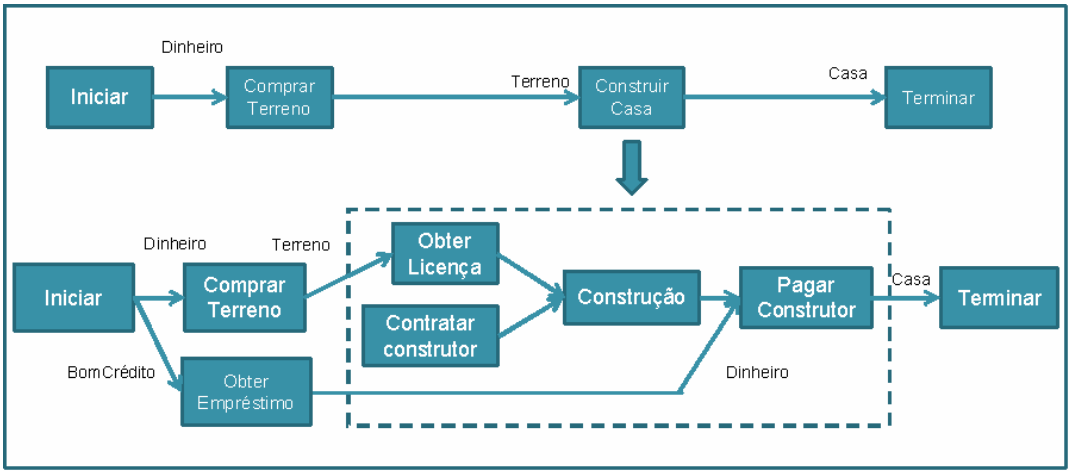
3. Conectar as ligações causais. Para todas as pré-condições externas, deve existir uma ou mais ligações causais de uma ação anterior para a ação da decomposição que consome aquele recurso. Da mesma forma, devem ser feitas as ligações causais relativas aos efeitos externos.

Além disso, algumas outras modificações são necessárias por causa da ocultação de informações da decomposição hierárquica. O planejamento de ordem parcial realiza backtrack ao atingir um conflito entre uma ação com uma ligação causal, e não pode ser ordenada antes ou depois desta. Com a decomposição hierárquica, alguns destes

conflitos podem ser resolvido, fazendo com que seja necessário evitar certar oportunidades de poda.

5.1. Exemplo de Decomposição

O seguinte exemplo demonstra os passos da decomposição:



1. A ação 'Construir Casa' foi substituída pela decomposição no retângulo pontilhado. Neste caso, não existem ações que possam ser utilizadas para realizar o sub-task sharing, então todas as ações da decomposição foram instanciadas.

2. Neste caso, não existem restrições de ordenação, mas pode-se notar que a ação 'Contratar Construtor' é independente de ordenação. Portanto, não necessariamente requer que ocorra após a ação 'Comprar Terreno', podendo ocorrer antes desta.

3. Para a pré-condição 'Terreno', foi feita a ligação causal entre 'Comprar Terreno' e 'Obter Licença'. Para o efeito 'Casa', foi feita a ligação causal entre 'Pagar Construtor' e 'Terminar'. Note que a

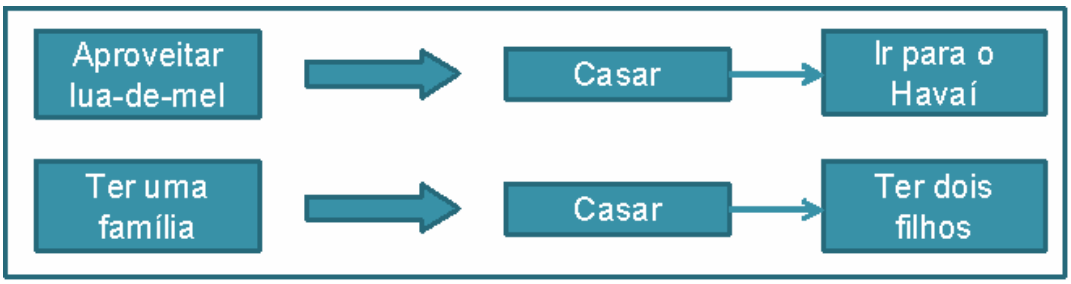
ação ‘Pagar Construtor’ possui outra pré-condição aberta ‘Dinheiro’, que foi suprida por outra ação, ‘Obter Empréstimo’.

5.2. Exemplo de Sub-task Sharing

O seguinte exemplo demonstra a importância do sub-task sharing:

Considere o objetivo 'Aproveitar lua-de-mel' e 'Ter uma família'. Utilizando a decomposição hierárquica, uma possível decomposição de 'Aproveitar lua-de-mel' poderia ser 'Casar' e 'Ir para o Havaí', enquanto que 'Ter uma família' poderia ser decomposta em 'Casar' e 'Ter dois filhos'.

Neste caso, ao se retirar o sub-task sharing, o planejamento HTN causaria a adição de duas ações de 'Casar', o que obviamente não é desejado.



5.3. Comparação

A seguir, faremos uma comparação entre o planejamento HTN puro e o planejamento híbrido HTN-POP, identificando problemas e possíveis soluções para eles.

5.3.1. HTN Puro

Problema:

O principal problema do planejamento HTN puro é que ele é um problema indecidível, mesmo sobre um espaço de estados finito.

Isso ocorre devido ao fato de que a decomposição de ações pode ser recursiva.

Solução:

- Eliminar a recursão, tendo em vista que poucos domínios requerem recursão.
- Limitar o comprimento das soluções. Desta forma, é possível controlar a procura sem ter grandes perdas .
- Adotar o planejamento híbrido POP HTN, pois o planejamento de ordem parcial por si só é capaz de decidir se um plano existe ou não.

5.3.2. Híbrido POP - HTN

Problema:

O planejamento de ordem parcial liga ações primitivas de maneira arbitrária. No planejamento híbrido, isto pode levar a soluções de difícil entendimento, perdendo a estrutura hierárquica do HTN.

Solução:

- Priorizar a decomposição sobre a adição de novas ações, mas tomando o cuidado de evitar a geração de longos planos HTN sem que sejam inseridas ações primitivas. Uma maneira de se fazer isso é utilizar uma função de custo que dá descontos para ações introduzidas através de decomposição.

5.4. Razões para se utilizar HTN

5.4.1. Custo

Uma das razões para se utilizar o planejamento HTN é em relação ao custo. Para um plano não-hierárquico com n ações e planejamento forward state-space:

- Forward state-space: custo $O(b^n)$
    - b: ações permitidas
    - n: número de ações

Por outro lado, o custo de plano de um planejamento HTN:

- HTN: custo $d^{(n-1)/(k-1)}$
    - d: possíveis decomposições dos não-primitivos
    - k: número de ações no próximo nível

Desse modo, podemos ver que o custo do planejamento HTN é menor desde que duas regras sejam obedecidas:

- A biblioteca de planos utilizada possui poucas decomposições
- As decomposições da biblioteca devem ser longas

5.4.2. Utilização prática

A grande maioria dos planejadores de aplicações de larga-escala são planejadores HTN, pois esse tipo de planejamento permite a utilização de um conhecimento humano especializado sobre como realizar tarefas de grande complexidade, utilizando pouco esforço computacional.

Um exemplo disso é o O-Plan, que combina o planejamento HTN com escalonamento, utilizado para desenvolver planos de produção

para a Hitachi. O O-Plan é capaz de resolver problemas envolvendo linhas de produção de 350 produtos diferentes, com 35 máquinas e mais de 2000 operações diferentes.

6. Conclusão

Muitas vezes os problemas reais são muito complexos para serem descritos por modelos simples, o que exige uma extensão do modelo. O planejamento com restrição de recursos e tempo é extremamente importante para problemas reais, como por exemplo, o planejamento do uso dos recursos de linha de produção visando minimizar o tempo de fabricação de cada produto, ou o caso do telescópio Hubble, que usa planejamento para otimizar o seu processo de observação.

Concluímos que o estudo dessa área da inteligência artificial é extremamente importante para problemas que apresentam recursos escassos, que correspondem a grande parte dos problemas do mundo real, para que tais recursos sejam aproveitados da melhor maneira.

7. Bibliografia

RUSSELL, Stuart J. ; NORVIG, Peter. *Artificial Intelligence, a Modern Approach*. 2nd Edition: Prentice Hall, 2003