UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

APLICANDO TÉCNICAS DE APRENDIZADO DE MÁQUINA EM PLANEJAMENTO

JEAN LUCAS DE SOUSA

Uberlândia - Minas Gerais
2014

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JEAN LUCAS DE SOUSA

APLICANDO TÉCNICAS DE APRENDIZADO DE MÁQUINA EM PLANEJAMENTO

Dissertação de Mestrado apresentada à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Inteligência Artificial.

Orientador:
Prof. Dr. Carlos Roberto Lopes

Uberlândia, Minas Gerais
2014

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU

S725a Sousa, Jean Lucas de, 1990-
2014 Aplicando as técnicas de aprendizado de máquina em planejamento / Jean Lucas de Sousa. -- 2014.
101 f. : il.
Orientador: Carlos Roberto Lopes.

Dissertação (mestrado) – Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1. Computação - Teses. 2. Aprendizado do computador - Teses. 3. Planejamento - Teses. I. Lopes, Carlos Roberto, 1962- II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Aplicando Técnicas de Aprendizado de Máquina em Planejamento**" por **Jean Lucas de Sousa** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 02 de junho de 2014

Orientador: _____
Prof. Dr. Carlos Roberto Lopes
Universidade Federal de Uberlândia

Banca Examinadora:

_____
Profª. Drª. Rita Maria da Silva Julia
Universidade Federal de Uberlândia

_____
Prof. Dr. Flávio Tonidandel
Centro Universitário da FEI

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: junho de 2014

Autor:	**Jean Lucas de Sousa**
Título:	**Aplicando Técnicas de Aprendizado de Máquina em Planejamento**
Faculdade:	**Faculdade de Computação**
Grau:	**Mestrado**

_____

Autor

Dedicatória

Aos meus pais Jorge e Edna e minha irmã Jorgiane.
A minha namorada Adriana.

Agradecimentos

Agradeço...

A minha família e namorada, que me apoiaram e acreditaram em mim...
Aos colegas, companheiros de viagens, amigos, que foram tantos, mas essenciais à realização dessa etapa...
Aos professores do PPGCC, principalmente Márcia, Gina, Rita e Dino, professores da IA, que compartilharam seus conhecimentos e me apresentaram muitas das técnicas utilizadas nesse trabalho...
Ao meu orientador Carlos, que me encaminhou sempre nas melhores direções para a pesquisa...
A quem participou direta ou indiretamente na produção dessa dissertação, meu muito obrigado!

"O ontem é história. O amanhã é mistério. Mas o hoje é uma dádiva, por isto se chama 'presente'."
(Adalberto Godoy)

Resumo

Em termos de abordagem clássica, sistemas de planejamento ou planejadores concentram-se em gerar automaticamente uma sequência de ações que transforma uma configuração (estado) inicial de objetos em outro estado em que um dado objetivo é satisfeito. Sistemas de planejamento foram utilizados para resolver uma variedade de problemas com sucesso. Apesar disso, nenhum planejador é melhor que todos os outros quando aplicados a problemas distintos.

O planejamento probabilístico é uma extensão do planejamento clássico que trabalha sobre um ambiente não determinístico. Assim como no planejamento clássico, diversos planejadores foram propostos para resolver problemas, porém nenhum planejador é capaz de superar totalmente os outros em todos os problemas.

Neste trabalho, descreve-se uma abordagem que consiste em extrair características do problema a ser resolvido e determinar, a partir de um conjunto de planejadores clássicos e probabilísticos, um que seja capaz de resolver o problema com eficiência. Em nossa abordagem, são utilizados algoritmos de aprendizado de máquina para determinar o melhor planejador dentre o portfólio que resolve o problema.

A seleção dos planejadores se mostrou eficiente nos testes tendo mostrado bons resultados nos experimentos ao superar os planejadores de portfólio que conseguiram os melhores resultados nas competições de planejamento em ambas as áreas (planejamento clássico e probabilístico).

Palavras chave: classificação, portfólio, aprendizado de máquina, planejamento clássico, planejamento probabilístico

Abstract

In terms of classical planning, planners objectives are generate a sequence of actions that converts an initial configuration (state) into another state that attends a goal. Planning systems have been used in solving a variety of problems with success. However, no planner is capable of outperforming all the others when applied to distinct problems.

Probabilistic planning is an extension of classical planning that works with stochastic environments. Just as in classical planning, several planners were proposed to solve probalistic planning problems. However, no planner is capable of outperform all others when applied to distinct problems.

In this work we describe our approach that is capable of extracting features of a planning problem and determining a classical or probabilistic planner from a portfolio that can solve the problem. We use machine learning algorithms to determine the best planner from the porfolio that solves a problem.

Our approach showed good results in the experiments. Our approach outperformed the best planners from a recent planning competition in both areas (classical and probabilistic planning).

Keywords: classification, portfolio, machine learning, classical planning, probabilistic planning

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

A*	*A star* - Algoritmo para busca de caminhos ótimos em grafos
ADD	*Algebraic Decision Diagram*
ADL	*Action Description Language*
AG	Algoritmo Genético
ARFF	*Attribute-relation File Format*
BDD	*Binary Decision Diagram*
BFS	*Best First Search*
CG	*Causal Graph*
CPF	*Conditional Probabilistic Function*
DER	Diagrama de entidade-relacionamento
DTG	*Domain-Transition Graph*
EHC	*Enforced Hill Climbing*
FF	*Fast Forward*
FD	*Fast Downward*
FDSS	*Fast Downward Stone Soup*
HAP	*Highly Adjustable Planner*
HAP-NN	*Highly Adjustable Planner Nearest Neighbor*
HC	*Hill Climbing*
HSP	*Heuristic Search Planner*
IA	Inteligência Artificial
IPC	*International Planning Competition*
IPPC	*International Probabilistic Planning Competition*
k-NN	*k Nearest Neighbor*
LRTA*	*Learning Real-Time A**
M&S	*Merge & Shrink*
MCT	*Monte-Carlo Tree*
MDP	*Markov Decision Process*
NB	Naïve Bayes
NFL	*No Free Lunch*
PbP	*Portfolio-based Planner*

PDDL	*Planning Domain Definition Language*
PPDDL	*Probabilistic Planning Domain Definition Language*
PG	*Planning Graph*
RDDL	*Relational Dynamic Influence Diagram Language*
RPG	*Relaxed Plannig Graph*
SOM	*Self Organizable Mapping*
STRIPS	*STanford Research Institute Problem Solver*
SVM	*Support Vector Machine*
UCM	*Upper Confidence Tree*
W-A*	*Weighted A star*
WEKA	*Waikato Environment for Knowledge Analysis*

Capítulo 1

Introdução

A evolução da informática é notável ao longo dos anos. E junto com essa evolução, os problemas também ficaram cada vez mais complexos de serem resolvidos. Visando reduzir esta complexidade, pesquisas vêm sendo desenvolvidas com o objetivo de entender como o ser humano raciocina e assim aplicar tal conhecimento no desenvolvimento de técnicas que possam auxiliar na solução de problemas. Em Ciência da Computação tais pesquisas concentram-se em uma área denominada Inteligência Artificial (IA). De uma forma geral, a Inteligência Artificial busca resolver problemas utilizando habilidades humanas sintetizadas em técnicas computacionais capazes de realizar tarefas automaticamente [Russell e Norvig 2009].

Dentre as diversas áreas de pesquisa em IA, a área de planejamento (*AI Planning*) é uma das mais intrigantes. Sistemas de planejamento ou planejadores concentram-se em gerar automaticamente uma sequência de ações que transformam uma configuração (estado) inicial de objetos em outro estado em que um dado objetivo é satisfeito. Existem diversas abordagens para implementar sistemas de planejamento. Tais abordagens se diferem na forma de se representar estados, ações e planos. Como exemplo de abordagens tem-se o planejamento clássico, planejamento condicional, planejamento probabilístico, entre outros. Nesta dissertação, a ênfase é dada às abordagens clássica e probabilística.

No planejamento clássico considera-se que os ambientes são classificados em totalmente observáveis (permitem que sensores obtenham todas as informações do estado); estáticos (se modificam apenas durante a execução de uma ação, nunca na etapa de decisão); e determinísticos (as ações sempre são realizadas com sucesso) [Russell e Norvig 2009]. O planejamento probabilístico é uma extensão do planejamento clássico que trabalha sobre um ambiente não determinístico, onde cada uma das ações possui uma probabilidade de ocorrer com sucesso ou não. Isso indica que nem sempre o estado corrente sofre as alterações esperadas decorrentes da execução de uma determinada ação.

Algoritmos de busca tem sido muito usados recentemente para resolver problemas de planejamento clássico. De fato, algoritmos de planejamento implementados com diferentes técnicas de busca tem se mostrado eficiente na solução de um grande número de

problemas. O *Fast Forward* [Hoffmann e Nebel 2001] é um exemplo de planejador de sucesso implementado usando o algoritmo de busca *Enforced Hill Climbing*. Apesar disso, nenhum algoritmo de planejamento é superior aos demais [Wolpert e Macready 1997]. Um determinado algoritmo de planejamento pode não resolver um problema já resolvido por outro algoritmo. A partir dessas observações, uma das tendências nessa área é selecionar o melhor algoritmo ou uma combinação dos melhores algoritmos para resolver um problema em particular [Kotthoff 2012]. Desta forma, passa-se a ter um problema de seleção de algoritmo a ser resolvido [Rice 1976]. Ao longo desta linha de raciocínio foram desenvolvidos os chamados portfólios de planejadores. Na criação de um portfólio, uma das grandes dificuldades ainda é determinar o melhor planejador para resolver uma instância de um determinado problema [Vallati e Kitchin 2012].

Assim como no planejamento clássico, em planejamento probabilístico diversos planejadores foram propostos para resolver os problemas, incentivados principalmente pela *International Probabilistic Planning Competition* (IPPC) - Competição Internacional de Planejamento Probabilístico. É possível notar que, apesar de dois planejadores se destacarem nos resultados finais (*PROST* [Keller e Eyerich 2012] e *Glutton* [Kolobov et al. 2012]), outros planejadores propostos conseguiram resultados positivos em problemas individuais, o que nos situa no mesmo problema enfrentado no planejamento clássico: nenhum planejador é capaz de superar totalmente os outros em todos os problemas [Vallati e Kitchin 2012].

Objetivando mitigar esta dificuldade em planejamento clássico e probabilístico, esta dissertação descreve a proposta de um portfólio de planejamento de propósito geral, independente de domínio, para ambas as áreas. A partir da especificação de um determinado problema de planejamento, levantam-se características que permitem determinar os melhores algoritmos para solução do problema. A escolha dos melhores planejadores é feita com base em algoritmos de aprendizado de máquina. Na escolha dos melhores considerou-se o tempo para a geração do plano em planejamento clássico e maximização da pontuação em planejamento probabilístico.

Analisando as formas de representação de um problema de planejamento é possível extrair características que são inerentes ao problema em questão, e que podem auxiliar a decidir que algoritmo de planejamento será usado para resolvê-lo. Como exemplo, podemos citar a quantidade de ações possíveis em um problema, ou o número de objetos pertencentes à ele, onde essas características podem ser extraídas de um arquivo PDDL (linguagem formal para descrição de um problema de planejamento) e estão diretamente relacionadas ao tamanho do problema. Em [Vrakas et al. 2003], foram utilizadas 35 características para definir um conjunto de regras e posteriormente os parâmetros que seu planejador utilizaria relacionados à direcionalidade da busca, heurística, relaxamento, entre outros. [Roberts e Howe 2007] levantou 32 características existentes na especificação de diversos problemas de planejamento na tentativa de selecionar um planejador

que resolveria o problema. [Alhossaini e Beck 2013] também extraiu características específicas dos domínios de alguns problemas tentando definir o melhor planejador para resolvê-lo. Inicialmente, em nossa proposta foram selecionadas 38 características distintas. Posteriormente, utilizando um algoritmo genético foi possível determinar as melhores características representativas do problema.

O uso de aprendizado de máquina na tentativa de classificar problemas já foi utilizada em planejamento. Vários portfólios de planejamento clássico usando técnicas de classificação foram propostos, como a utilização de técnicas como o algoritmo C4.5 e o k-NN, partindo de características extraídas do problema, na tentativa de predizer se um planejador teria ou não sucesso ao tentar encontrar um plano para um determinado problema [Roberts e Howe 2007, Cenamor et al. 2012, Alhossaini e Beck 2013].

Essa dissertação propõe a utilização de técnicas de classificação em uma abordagem semelhante aos trabalhos de Vrakas [Vrakas et al. 2003] e Roberts [Roberts e Howe 2007] para planejamento clássico, utilizando um refinamento das características propostas, uma atualização dos planejadores utilizados e uma mudança de objetivo: enquanto em Vrakas a classificação é usada para parametrizar seu planejador, e em Roberts para determinar o sucesso de um planejador, nosso objetivo é determinar qual o planejador é capaz de resolver um problema no menor tempo possível, no intuito de credenciar nosso planejador para participar de competições de planejamento. Posteriormente, partiu-se para a criação de um portfólio de planejamento, combinando os melhores classificadores em um único planejador, com o objetivo de refinar os resultados com relação à taxa de sucesso. O procedimento seguido para planejamento clássico foi também aplicado ao planejamento probabilístico. Nessa área não existe abordagem semelhante à aplicada nesse trabalho.

A proposta realizada neste trabalho se mostrou eficiente ao determinar os melhores planejadores para os problemas nos testes realizados. Desta forma foi possível superar o desempenho dos planejadores que anteriormente haviam conseguido os melhores resultados nas competições de planejamento considerando-se as trilhas de planejamento clássico e probabilístico.

1.1 Contribuição

As principais contribuições deste trabalho são:

- implementação de um portfólio de planejadores baseado em classificação na área de planejamento clássico, capaz de determinar os planejadores com maior probabilidade de sucesso para um problema e que solucionarão o mesmo no menor tempo possível;
- geração de um classificador eficiente na área de planejamento probabilístico, capaz de determinar o melhor planejador para resolver um determinado problema;

1.2 Organização da Dissertação

O Capítulo 2 apresenta conceitos básicos de algoritmos busca, planejamento em Inteligência Artificial, classificação e trabalhos relacionados ao contexto da dissertação.

O Capítulo 3 apresenta detalhes da arquitetura para geração e treinamento dos classificadores que participarão dos portfólios de planejamento.

O Capítulo 4 descreve os resultados detalhados encontrados pelos classificadores gerados comparando com os planejadores com melhores resultados na área de planejamento clássico.

O Capítulo 5 descreve os resultados detalhados encontrados dos classificadores gerados, porém na área de planejamento probabilístico.

Por fim, o Capítulo 6 conclui o trabalho trazendo perspectivas de trabalhos futuros.

Capítulo 2

Fundamentos Teóricos e Trabalhos Correlatos

Neste capítulo são discutidos os fundamentos teóricos e trabalhos correlatos, para melhorar a compreensão do trabalho desenvolvido e exibir os trabalhos importantes já desenvolvidos com aspectos semelhantes. Este capítulo foi estruturado da seguinte forma: na Seção 2.1 são descritos conceitos e algoritmos de busca relacionados ao tema; na Seção 2.2 são apresentados conceitos e alguns sistemas de planejamento, além de aspectos relacionados aos portfólios de planejadores; e na Seção 2.3 são apresentadas técnicas de classificação e suas aplicações na área de planejamento.

2.1 Busca

Um problema de busca consiste em determinar uma possível sequência de transições entre estados, representados numa estrutura chamada árvore de busca, que é capaz de sair de um nó inicial e chegar a um nó meta. De maneira geral, qualquer problema pode ser formulado como uma árvore de busca, onde cada nó caracteriza uma determinada situação do problema, chamada estado, e as arestas direcionadas são ações a serem aplicadas no estado origem que o alteram para o estado destino. Um estado é a representação formal de como o ambiente está durante um determinado instante de tempo [Russell e Norvig 2009].

Um algoritmo de busca tem como objetivo encontrar uma solução para uma determinada tarefa que lhe é passada. Todo problema sempre necessita da informação do nó inicial que a busca terá como base, chamado estado inicial, e do objetivo a ser solucionado, que é meta do problema. A solução de um problema de busca é composta por uma sequência de estados, sendo que cada estado representa parte da solução [Russell e Norvig 2009].

Como existem diversos algoritmos de busca, são necessários critérios para definir a

escolha em cada problema específico. [Russell e Norvig 2009] cita que as avaliações de algoritmos de busca são geralmente feitas com base em quatro critérios:

- **Completude**: propriedade do algoritmo encontrar uma solução sempre que alguma exista;
- **Otimalidade**: o algoritmo sempre encontra a melhor solução para o problema;
- **Complexidade de tempo**: tempo médio de execução do algoritmo;
- **Complexidade de espaço**: consumo de memória do algoritmo;

Um algoritmo de busca pode ser caracterizado de acordo com as informações que ele utiliza para explorar entre estados. Em função disto, ele pode ser caracterizado como um algoritmo de busca cega ou um algoritmo de busca informada. Uma busca cega realiza todas as avaliações do estado fazendo uso apenas de informações inerentes ao problema, como custos de caminho. Tudo que se pode fazer é expandir um estado, ou seja, gerar estados sucessores a partir de um outro estado qualquer, e distinguir um estado meta de um não-meta. Em uma busca informada, existe a adição de uma função heurística, que estima a distância entre o estado atual e a meta, para auxiliar a definir a ordem que os sucessores serão visitados. Uma heurística $h(s)$ serve para estimar a distância de um estado s até a meta, onde algoritmos de busca priorizam por expandir estados que tem menor valor de h, ou seja, estados que acredita-se estarem mais próximos à meta para que a busca seja executada o mais rápido possível. Em geral, algoritmos de busca cega possuem tempo de execução menor, enquanto algoritmos de busca informada têm características de completude e otimalidade [Edelkamp e Schrödl 2012].

Os algoritmos de busca também podem ser caracterizados de acordo com o procedimento de exploração dos estados, sendo classificados como algoritmos de busca sistemática ou algoritmos de busca local. Uma busca sistemática é realizada armazenando todos os possíveis sucessores de cada nó para que posteriormente sejam visitados. A busca sistemática garante a completude do problema, pois se em algum instante de tempo nenhum estado é passível de ser visitado, é possível afirmar que não existe solução, e se em algum momento um estado for compatível com a meta, este será a solução do problema. A busca local, diferentemente da busca sistemática, possui informação de apenas um estado por vez, e em cada escolha de sucessor o estado anterior é descartado e um novo estado é tomado como base. Algoritmos de busca local geralmente possuem implementação simples, porém não garantem completude do problema, pois podem ficar presos em ótimos locais[1] [Hoos e Sttzle 2004].

A estratégia de um algoritmo de busca varia de acordo com cada algoritmo, porém o ponto de partida define se o algoritmo é do tipo *forward* ou *backward*. Um algoritmo de busca do tipo *forward* realiza a sua busca a partir do estado inicial e caminhando pelos

[1]Soluções que são as melhores entre os vizinhos, mas não são a melhor solução para o problema

estados subsequentes até encontrar um estado que satisfaça a meta. Já uma busca do tipo *backward* realiza o processo contrário: inicia a partir de um estado que satisfaça a meta e realiza a busca procurando pelo estado inicial. Uma dificuldade ao executar uma busca *backward* é garantir que apenas um estado satisfaça a meta, pois em alguns problemas vários estados podem atendê-la, então existiriam vários estados que a busca *backward* poderia partir. Geralmente, os algoritmos de busca são implementados como *forward*, mas a estratégia *backward* pode ser útil para problemas que geram muitos sucessores nos estados iniciais [Malik et al. 2004].

Por último, um algoritmo pode ser *off-line* ou *on-line*. Nos algoritmos *off-line*, a fase deliberativa antecede totalmente a fase de execução, o que permite que todo tempo e espaço seja alocado para encontrar a solução, e após isso, a mesma é executada. Os algoritmos *on-line*, também denominados algoritmos de busca em tempo real, alternam as fases de deliberação e execução durante o processo de busca. Um algoritmo *on-line* não conhece toda a informação sobre o espaço de busca, e após um certo tempo é necessário que ele retorne uma solução. De um modo geral, algoritmos *off-line* tem um tempo de resposta maior que os *on-line*, e conseguem encontrar soluções melhores pelo maior conhecimento sobre o problema [Furcy 2004].

2.1.1 Algoritmos de Busca

Com a evolução das heurísticas, algoritmos de busca cega se tornaram pouco efetivos em problemas com alta complexidade. Dentre os principais algoritmos de busca cega, pode-se citar:

- **Busca em largura**: onde o estado inicial é expandido primeiro, e depois seus sucessores, os sucessores de seus sucessores, e assim por diante, até que se encontre a meta;
- **Busca em profundidade**: os estados expandidos nessa estratégia são armazenados em uma fila, onde ao serem gerados são colocados na cabeça da fila, e o estado a ser expandido é retirado também da cabeça da fila. Entre suas variações pode-se citar busca em profundidade limitada e busca em profundidade iterativa;
- **Busca de custo uniforme**: seleciona os sucessores de acordo com uma função de avaliação $f(n)$ a partir do custo para expandir o estado ($g(n)$), i.e. $f(n) = g(n)$;

A busca heurística utiliza um conhecimento específico do problema na escolha do próximo estado a ser expandido. Geralmente buscas informadas são baseadas em estratégias cegas, adicionando a informação heurística para auxílio da avaliação.

O algoritmo BFS (*Best-First Search*) é um algoritmo de busca sistemática e informada *off-line*, baseado na busca de custo uniforme, que seleciona os sucessores de acordo com uma função de avaliação $f(n)$ que corresponde a sua estimativa de distância (função

heurística $h(n)$), i.e. $f(n) = h(n)$. É mantida uma lista fechada (*closed*), que armazena os estados já visitados, e uma fila de prioridade ordenada pelo menor valor $f(n)$ (*open*), que armazena os estados que estão disponíveis para serem visitados. A cada iteração do algoritmo, o estado presente na lista aberta com menor $f(n)$ é selecionado, e se esse não for a meta, todos os sucessores que não pertençam à lista fechada e à lista aberta são adicionados para serem visitados posteriormente. O algoritmo para com sucesso quando o estado selecionado da lista aberta é a meta e para com falha quando a lista aberta fica vazia [LaValle 2006]. Seu pseudo-código está descrito no Algoritmo 1:

Algoritmo 1 BFS($s_{inicial}, s_{meta}$)

```
1: open ← s_inicial
2: closed ← ∅
3: enquanto open ≠ ∅ faça
4:    s_current ← open.remove()
5:    se s_current = s_meta então
6:       retorna sucesso
7:    fim se
8:    closed.add(s_current)
9:    open.add(sucessores(s_current))
10: fim enquanto
11: retorna falha
```

A principal vantagem do BFS é a completude, que garante que sempre a solução será encontrada. Como desvantagem, o uso de heurísticas como única função avaliativa o torna dependente da efetividade da estimativa: quanto melhor sua função heurística, menor será o tempo necessário para que o BFS encontre a solução.

O algoritmo A* (*a star*) também é um algoritmo de busca completo, executado de forma sistemática e informada, que garante sempre encontrar a solução ótima caso a heurística fornecida seja admissível[2], para busca em árvore, ou consistente[3], para busca em grafo. O funcionamento do A* é semelhante ao do BFS, porém com uma modificação na função de avaliação $f(n)$: ela é formada pela soma dos elementos $g(n)$ e $h(n)$, onde $g(n)$ é o custo do caminho a partir do estado inicial até o estado atual e o $h(n)$ é a função heurística, i.e. $f(n) = g(n) + h(n)$. Outra modificação é a seleção dos sucessores: um sucessor pode ser escolhido mesmo que pertença à lista fechada, caso o $f(n)$ atual seja menor que o $f(n)$ armazenado na lista fechada. Por último, ao adicionar sucessores na lista aberta ele verifica se o nó atual já pertence a lista aberta. Se não pertencer, ele é adicionado. Se ele já pertence a lista aberta, é comparado o $f(n)$ atual com o $f(n)$ armazenado na lista aberta. Se o $f(n)$ atual for menor, o nó é substituído na lista aberta [Hart et al. 1968].

[2]Uma heurística admissível é uma heurística onde a estimativa de custo fornecida para um estado qualquer até a meta nunca supera seu custo real ($h(n) < g(n)$).

[3]Uma heurística é consistente se para cada estado s, e seus sucessores s' gerados por uma ação a, o custo estimado para chegar na meta a partir de s não é maior que o custo da ação a mais o custo estimado para chegar à meta a partir de s' ($h(s) \leq h(s') + c(s, a, s')$)

Uma variação desse algoritmo que geralmente acelera o processo de busca é chamada *Weighted* A* ou W-A*. O algoritmo W-A* tem o mesmo comportamento que o algoritmo A* tradicional, porém usa um parâmetro ε ($\varepsilon > 1$) definido antes de sua execução que pondera a heurística a ser usada. Portanto, a nova função avaliação é $f(n) = g(n) + \varepsilon * h(n)$. O incremento ao valor heurístico pode acelerar o processo mas pode tornar a heurística inadmissível [Pearl 1984].

O algoritmo *Learning Real Time A** (LRTA*) [Korf 1990] é outra variação do A*, porém é um algoritmo de busca local e com característica *on-line*: a heurística tem seus valores atualizados durante a busca, permitindo que o algoritmo possa aprender informações positivas ou negativas a respeito dos estados posteriores a serem expandidos. O LRTA* parte do estado inicial se deslocando para o sucessor mais promissor de acordo com sua função de avaliação ($f(n) = g(n) + h(n)$). A busca é concluída ao atingir a meta, porém, a cada iteração, o algoritmo atualiza a estimativa da distância até a meta. Para se chegar à solução ótima, é necessário executar o algoritmo mais de uma vez, até que não ocorram mais atualizações nas heurísticas.

Outro algoritmo de busca com característica local é o *Hill Climbing* (HC) [Russell e Norvig 2009]. O HC é um algoritmo que funciona como se segue: seleciona-se o estado inicial como estado corrente. A partir do estado corrente, o primeiro sucessor encontrado com menor $h(n)$ ($h(succ) < h(current)$) é selecionado e se torna o novo estado corrente. A seleção se repete até que nenhum estado com menor $h(n)$ possa ser selecionado, resultando em falha, ou que um estado que atenda a meta seja selecionado, resultando em sucesso na busca. O HC geralmente é mais efetivo em problemas com tempo restrito para se executar a busca. A principal deficiência do algoritmo HC é que o mesmo pode entrar em um estado ótimo local, ocasionando ou falha na busca, ou resultando em uma solução não-ótima. Existem variações propostas para evitar os ótimos locais, como o *Steepest Ascent Hill Climbing* [Coppin 2004] que avalia todos os sucessores primeiro, e seleciona dentre todos o com menor $h(n)$, ou o *Random-Restart Hill Climbing*, que seleciona um novo estado inicial aleatoriamente ao encontrar um ótimo local.

Outra variação é o *Enforced Hill Climbing* (EHC) [Hoffmann e Nebel 2001], que é modificado na etapa de seleção de sucessores: uma busca em largura é executada a partir do estado corrente, permitindo que sucessores de níveis inferiores também possam ser acessados. Então, se nenhum sucessor direto possui valor $h(n)$ para assumir a posição do estado corrente, os sucessores desses sucessores também serão testados em largura, e o processo é repetido até que exista um sucessor com $h(n)$ menor ou que todos os sucessores de todos os níveis sejam testados para um determinado estado. O EHC está descrito no Algoritmo 2:

A lista aberta (*open*) na busca em largura (linhas 5 a 15) assume a estrutura de dados fila, onde os estados a serem adicionados primeiro sempre serão removidos primeiro. Quando um estado com menor valor é encontrado, ele se torna o novo sucessor (linhas 8

Algoritmo 2 EHC($s_{inicial}, s_{meta}$)

```
1:  s_current ← s_inicial
2:  open ← ∅
3:  enquanto s_current ≠ s_meta faça
4:      melhorValor ← h(s_current)
5:      open.add(sucessores(s_current))
6:      enquanto open ≠ ∅ faça
7:          s_succ ← open.remove()
8:          se h(s_succ) < h(s_current) então
9:              s_current ← s_succ
10:             open ← ∅
11:             break
12:         senão
13:             open.add(sucessores(s_succ))
14:         fim se
15:     fim enquanto
16:     se h(s_current) = melhorValor então
17:         retorna falha
18:     fim se
19: fim enquanto
20: retorna sucesso
```

a 11) e a busca em largura é interrompida. Se ao concluir a busca em largura nenhum estado com valor de heurística melhor for encontrado (linhas 16 a 18), a busca para com falha. Se em algum momento, um dos sucessores atender a meta, a busca é interrompida e retorna sucesso.

Os Algoritmos Genéticos (AG) são métodos de busca baseado na abstração da teoria da evolução natural, onde cada possível solução para um determinado problema é a simulação de um cromossomo. Esses cromossomos evoluem durante as gerações através de métodos de seleção natural e operadores genéticos, como o *crossover* e a mutação. A principal inspiração de um AG é a ideia de que boas soluções podem ser combinadas reaproveitando parte da informação que a torna eficiente para gerar novas soluções ainda melhores do que a original [Mitchell 1998].

O fluxograma do AG da Figura 2.1, adaptado de [Goldberg 1989], descreve o fluxo de funcionamento dos AGs convencionais. As etapas estão descritas posteriormente:

- **Gerar a população inicial**: são gerados vários indivíduos, cada um com uma possível solução distinta para um determinado problema.
- **Avaliação dos indivíduos**: cada indivíduo é avaliado a partir da reação de sua solução com o problema. Os indivíduos recebem uma nota, que é o critério para informar se uma solução é boa ou ruim.
- **Seleção dos pais e cruzamento**: são selecionados pares de indivíduos, geralmente por um critério que leva em consideração a avaliação. Cada par de indivíduos fará

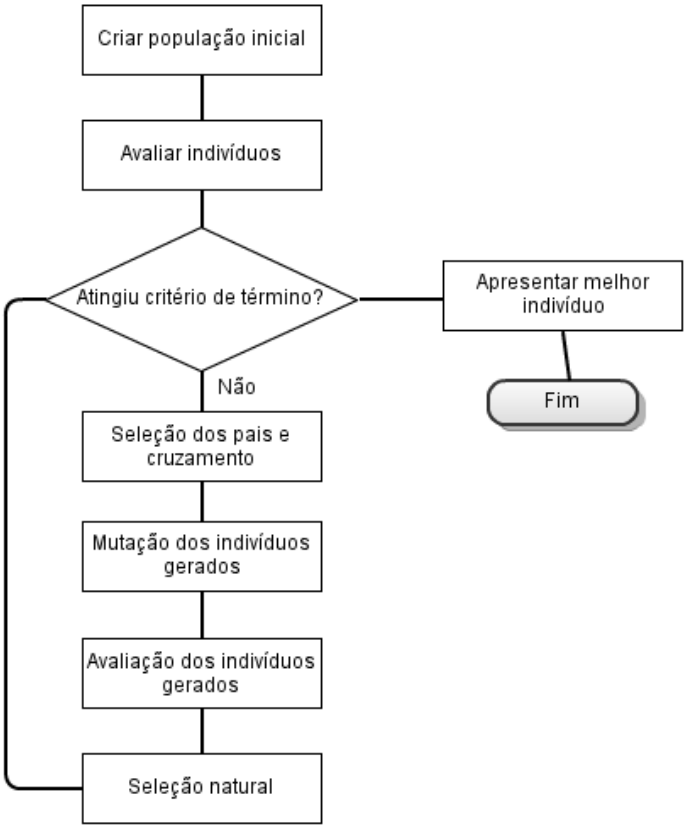

Figura 2.1: Fluxograma do Algoritmo Genético

parte de um cruzamento (*crossover*) gerando novos indivíduos com características semelhantes aos pais.

- **Mutação dos indivíduos gerados**: cada indivíduo gerado no cruzamento pode sofrer uma pequena perturbação em suas informações, na tentativa de melhorar as informações armazenadas pelo indivíduo.
- **Seleção natural**: a população de indivíduos deve voltar ao tamanho inicial, para isso alguns indivíduos devem ser descartados. Geralmente, os indivíduos com melhor avaliação são mantidos entre as gerações.

Os AGs são utilizados para resolver problemas computacionais, geralmente problemas de otimização combinatória, buscando soluções pseudo-ótimas, pois apresentam ideias baseadas na natureza que trazem boa convergência aos problemas que tentam resolver. Isso se deve principalmente a utilização dos operadores genéticos de maneira correta, que são capazes de combinar duas boas soluções para gerar outra ainda melhor [Mitchell 1998].

2.2 Planejamento

Planejamento é um ramo de pesquisa da Inteligência Artificial (IA), que tem como objetivo obter uma sequência de ações a serem aplicadas a partir de um estado inicial com a finalidade de atingir uma determinada meta, que pode ser um ou mais estados, que resolve o problema [Russell e Norvig 2009].

Um plano é uma sequência de ações que, ao aplicadas em um estado inicial, o alteram de forma que ao fim da aplicação dessas ações atenda a meta do problema atual. As transições entre estados intermediários são o resultado da execução de cada ação, que consome recursos existentes e produz novos recursos. Um sistema que gera planos para um problema é chamado planejador [Malik et al. 2004].

Um problema de planejamento geralmente é definido a partir da especificação do estado inicial do problema, da meta a ser satisfeita e das ações que podem ser aplicadas a um estado do problema. Uma ação é caracterizada por pré-condições e efeitos. Quando um estado do problema satisfaz as pré-condições de uma ação isto torna possível sua execução resultando num estado em que os efeitos associados a ela se tornam verdadeiros [Weld 1999]. Formalmente, sua definição é composta por uma 5-upla [Branquinho 2009]:

1. S: conjunto finito dos estados;
2. $i \in S$: estado inicial;
3. A: conjunto finito de ações;
4. g: meta;
5. P: solução do problema, ou plano;

No planejamento, um estado é representado por um conjunto de proposições verdadeiras $\mathcal{P}$. Todas as proposições não informadas durante o estado são consideradas falsas. Uma ação α é representada por uma tupla $(pre(\alpha), add(\alpha), del(\alpha))$, onde $pre(\alpha)$ é um conjunto de pré-condições: proposições que devem ser verdadeiras para que a ação α execute, ou seja, a ação α só é aplicada em $\mathcal{P}$ se e somente se $pre(\alpha) \subseteq \mathcal{P}$; $add(\alpha)$ é um conjunto de proposições que passarão a ser verdadeiras no estado após a execução da ação α; e $del(\alpha)$ é o conjunto de proposições que passaram a ser falsas após a execução da ação α. Após a aplicação das ações, o novo estado $\mathcal{P}$ é representado por $\mathcal{P} - del(\alpha) \cup add(\alpha)$.

Na Figura 2.2, o problema do Mundo dos Blocos é representado: o objetivo é mover os blocos e posicioná-los de acordo com a ordem desejada. Os estados são representados por uma conjunção de predicados de primeira ordem, onde *Sobre(A, X)* diz que um bloco A está sobre X, sendo X um bloco ou a própria mesa, e *Livre(B)* diz que não há nada sobre o bloco B. As ações disponíveis são *Mover* e *MoverParaMesa*. Os estados inicial e meta são representados graficamente pela Figura 2.2. Suas descrições na forma lógica são dadas abaixo:

- **Estado Inicial:** *Sobre(A, mesa)* $\wedge$ *Sobre(B, mesa)* $\wedge$ *Sobre(C, mesa)* $\wedge$ *Bloco(A)* $\wedge$ *Bloco(B)* $\wedge$ *Bloco(C)* $\wedge$ *Livre(A)* $\wedge$ *Livre(B)* $\wedge$ *Livre(C)* $\wedge$ *Livre(mesa).*
- **Estado Meta:** *Sobre(A, B)* $\wedge$ *Sobre(B, C)* $\wedge$ *Sobre(C, mesa).*

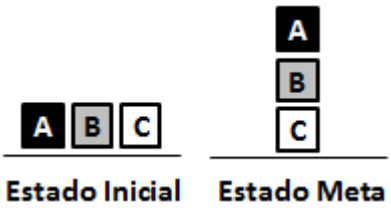

Figura 2.2: Exemplo: instância do problema Mundo dos Blocos

Pode-se usar além das proposições que definem a situação atual do estado (*Sobre(A,mesa), Sobre(B,mesa), Sobre(C,mesa), Livre(A), Livre(B), Livre(C), Livre(mesa)*) também proposições que garantem os tipos dos objetos a serem manipulados, como é o caso das proposições *Bloco(A), Bloco(B)* e *Bloco(C)*, que determinam que os objetos A, B e C são do tipo *Bloco*. A classificação dos objetos em tipos auxilia a evitar redundância de ações e limita as ações para os objetos, a partir de pré-condições.

Como já foi citado, as ações são operadores que mudam um dado estado, e são compostas por pré-condição, que define quando a ação pode ser aplicada a um estado, e efeito, que adiciona e/ou remove características de um estado, gerando assim um novo estado. No problema em questão tem-se duas ações: A ação *Mover(A, X, Y)* move um bloco A que está sobre X para Y, se ambos A e Y estão livres. As pré-condições *Bloco(A)* e *Bloco(Y)* são responsáveis por exigir que cada elemento dado como entrada seja um bloco, eliminando a possibilidade de chamar ações do tipo *Mover(A, X, mesa)*, pois *mesa* não é um *Bloco*. As pré-condições $A \neq X$, $A \neq Y$ e $X \neq Y$, determinam que os elementos sejam diferentes, para evitar ações redundantes. Depois que o movimento é feito, X fica livre e Y se torna ocupado. A ação *MoverParaMesa(A, X)* move um bloco A que está sobre X para a *mesa*. Essas ações são descritas abaixo:

- **Mover(b, x, y):**
  - **pré-condições:** *Sobre(b, x)* $\wedge$ *Livre(b)* $\wedge$ *Livre(y)* $\wedge$ *Bloco(b)* $\wedge$ *Bloco(y)* $\wedge$ *(b* $\neq$ *x)* $\wedge$ *(b* $\neq$ *y)* $\wedge$ *(x* $\neq$ *y)*
  - **efeitos:** *Sobre(b, y)* $\wedge$ *Livre(x)* $\wedge\neg$ *Sobre(b, x)* $\wedge\neg$ *Livre(y)*
- **MoverParaMesa(b, x):**
  - **pré-condições:** *Sobre(b, x)* $\wedge$ *Livre(b)* $\wedge$ *Bloco(b)* $\wedge$ *Bloco(x)* $\wedge$ *(b* $\neq$ *x)*
  - **efeitos:** *Sobre(b, mesa)* $\wedge$ *Livre(x)* $\wedge\neg$ *Sobre(b, x)*

Tendo as ações e informações a respeito do estado inicial, o mesmo é expandido gerando todos os sucessores possíveis, onde cada sucessor corresponde à aplicação de uma ação válida (que atende às pré-condições) para o estado atual. A Figura 2.3 mostra passo-a-passo a geração de um novo estado ao se executar a ação *Mover(A, mesa, B)* sobre estado inicial.

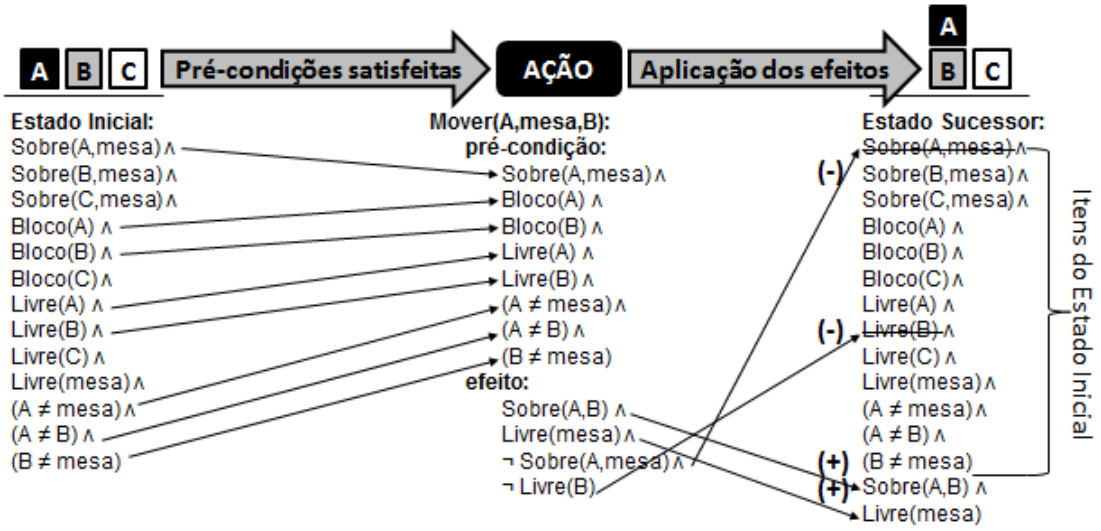

Figura 2.3: Exemplo: aplicando uma ação sobre o estado inicial

Para encontrar a solução de um problema, um planejador pode realizar sucessivas expansões a partir do estado inicial até encontrar o estado meta, guardando as ações usadas no caminho, construindo assim o plano (Figura 2.4).

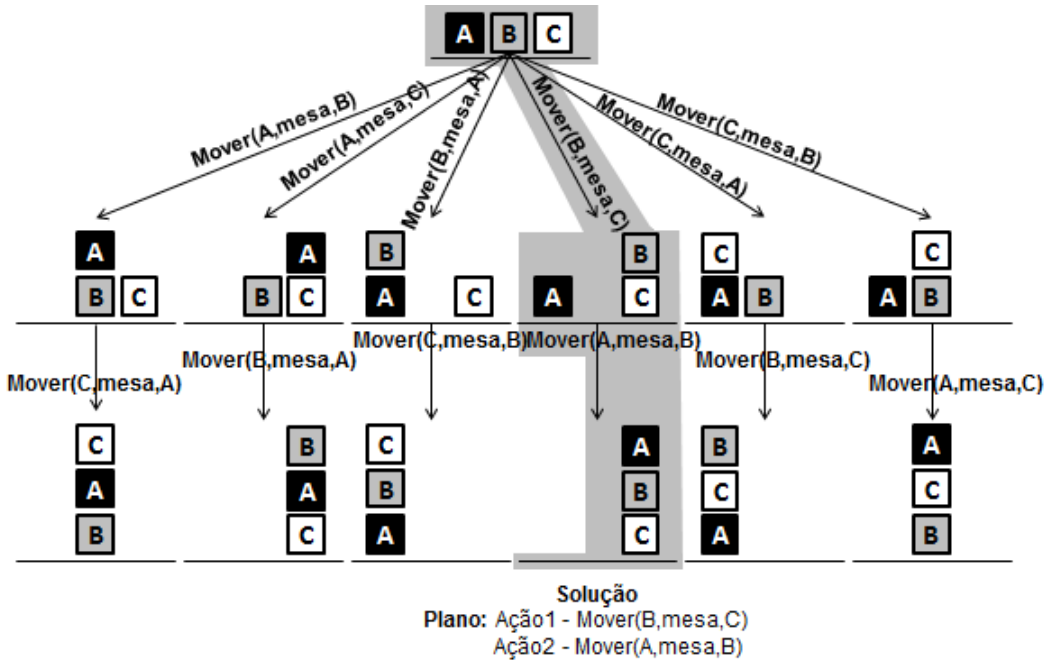

Figura 2.4: Exemplo: sucessivas expansões até encontrar a meta

No planejamento clássico, que é a abordagem principal desse trabalho, os ambientes onde o problema se situa sempre são totalmente observáveis (permitem que sensores obtenham todas as informações do estado); estáticos (se modificam apenas durante a execução de uma ação, nunca na etapa de decisão); e determinísticos (as ações sempre são realizadas com sucesso).

Por se assemelhar com um problema de busca, técnicas de busca são bastante utilizadas para resolver problemas de planejamento. Um problema de busca é solucionado com uma sequência de estados que representam a transição do estado inicial até a meta. Um problema de planejamento é solucionado com a sequência de ações que realizam a transição entre os estados.

2.2.1 Formalização de Problemas

A adoção de um formalismo comum para descrever domínios de planejamento promove uma maior reutilização das pesquisas e permite a comparação mais direta de sistemas e abordagens de planejamento, suportando, portanto, um progresso mais rápido no campo. A representação de problemas de planejamento deve ser feita por uma linguagem que seja eficiente para que algoritmos possam operar sobre o problema e que possa ser utilizada para representar diversos tipos de problemas [Russell e Norvig 2009].

A primeira linguagem formal para descrever domínios e problemas de planejamento foi a linguagem STRIPS (*STanford Research Institute Problem Solver*) [Fikes e Nilsson 1971]. Ela representa um estado como um conjunto de átomos, e as ações como operadores as quais possuem pré-condições que definem a aplicabilidade das ações e efeitos, que adicionam ou eliminam átomos de um estado para gerar estados sucessores. A linguagem STRIPS é muito semelhante à representação por proposições.

A linguagem ADL (*Action Description Language*) [Pednault 1989] foi uma melhoria a partir da linguagem STRIPS que permitia a representação de efeitos condicionais e quantificadores lógicos. Na ADL, as ações eram representadas por esquemas, ou seja, operadores cujos objetos são identificados por variáveis.

A linguagem PDDL (*Planning Domain Definition Language*) [Mcdermott et al. 1998] foi projetada para ser uma especificação neutra dos problemas de planejamento. Problemas e domínios são descritos através de um formalismo comum, e devido à sua generalização, qualquer planejador com capacidade de interpretação PDDL é capaz de usar o problema definido. A PDDL foi inspirada por outras notações e formalismos, como o ADL e STRIPS, tanto que problemas do tipo STRIPS e ADL podem ser descritos através do formalismo da linguagem PDDL.

Em PDDL, uma tarefa de planejamento é composta por: objetos, predicados ou preposições, estados e ações. Essas tarefas são separadas em dois arquivos: arquivo de domínio, que descreve predicados, ações e classes de objetos; e arquivo de problema, que descreve

objetos, estado inicial e especificação da meta do problema. O arquivo de domínio geralmente é único, permitindo diversos arquivos de problemas para um mesmo domínio, alterando os objetos a serem trabalhados e especificações iniciais e da meta. Com isso, para um mesmo domínio, é possível encontrar problemas fáceis e difíceis de se resolver.

Um arquivo de domínio tem a seguinte estrutura:

```
(define (domain <nome do domínio>)
  <código PDDL para predicados>
  <código PDDL para a 1ª ação>
  ...
  <código PDDL para a última ação>
)
```

Sendo `<nome do domínio>` um texto que identifica o domínio de planejamento. Já a estrutura de um arquivo de problema pode ser descrita como:

```
(define (problem <nome do problema>)
  (:domain <nome do domínio>)
  <código PDDL para objetos>
  <código PDDL para o estado inicial>
  <código PDDL para a especificação da meta>
)
```

Sendo `<nome do problema>` um texto que identifica um problema de planejamento, e `<nome do domínio>` nome do arquivo de domínio ao qual o problema se refere.

Inúmeras versões da PDDL foram propostas, dentre elas:.

- **PDDL 1.2** [Mcdermott et al. 1998]: Separação de domínios e problemas de planejamento;
- **PDDL 2.1** [Fox e Long 2003]: Adição de fluentes numéricos (quantidade, tempo, peso, distância), métricas de planos para avaliação qualitativa e ações contínuas onde pode-se ter variáveis, cumprimentos não-discretos, condições e efeitos;
- **PDDL 2.2** [Edelkamp e Hoffmann 2003]: Proposição de predicados derivados para modelar dependências entre fatos, e literais iniciais temporizados para modelar eventos que podem ocorrer em um determinado momento, independentemente da execução do plano;
- **PDDL 3.0** [Gerevini e Long 2005]: Introduziu o conceito de restrições de trajetória de estados, que se pesadas devem ser satisfeitas durante a execução de um plano para que ele seja considerado uma solução, e se leves são consideradas apenas como métricas;
- **PDDL 3.1**: Adicionou os objetos fluentes, onde intervalos não são descritos apenas com números, mas também com tipos de objetos do problema.

Juntamente com a representação formal dos problemas, a representação em grafo também é bastante utilizada em planejamento. As representações em grafo são utilizadas principalmente para tentar reduzir o problema, facilitar sua resolução ou extrair heurísticas dos problemas de planejamento.

Um grafo de planejamento [Blum e Furst 1995] (*Planning Graph* - PG) consiste em uma sequência de períodos de tempo no plano, chamados níveis. Um nível consiste em um conjunto de literais e um conjunto de ações, onde os literais representados em um nível são todos que podem ser verdadeiros naquele instante de tempo, e as ações são todas em que as pré-condições são atendidas. Literais não modificados são interligados por ações de persistência, e literais ou ações que geram conflitos entre si são interligados por vínculos de exclusão mútua (*mutex*). A construção do grafo de planejamento é realizada em tempo polinomial em relação ao número de ações e proposições de um grafo. Durante a criação de um grafo de planejamento, novos níveis são gerados até que o subsequente seja idêntico ao anterior. A detecção de *mutexes* é feita inicialmente para as ações, usando uma das três regras:

- **Efeitos inconsistentes:** uma ação nega o efeito da outra;
- **Interferência:** uma ação nega uma pré-condição de outra;
- **Necessidades conflitantes:** a pré-condição de uma ação possui *mutex* com a pré-condição de outra ação.

Após a atribuição dos *mutexes* entre ações, é necessário atribuir as exclusividades entre literais. Um literal é exclusivo com outro se um nega o efeito do outro, ou se todas as ações que geram o primeiro literal possuem *mutex* com todas as ações que geram o segundo literal. A Figura 2.5 apresenta um exemplo de grafo de planejamento para um problema simples: ter bolo e comer bolo também.

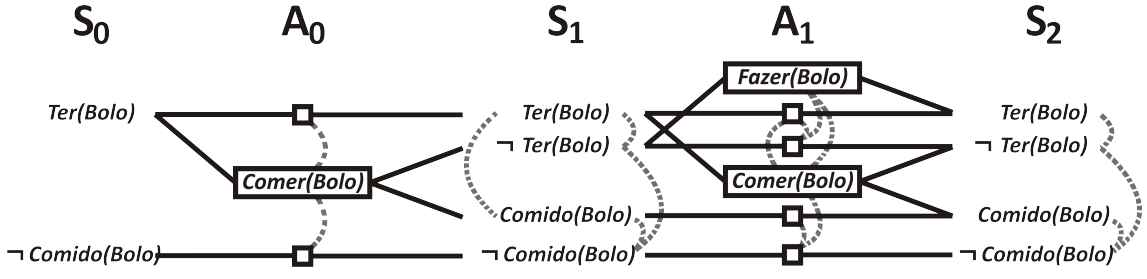

Figura 2.5: Grafo de planejamento: ter bolo e comer bolo também

No exemplo acima, o estado inicial é composto pelo literal *Ter(Bolo)*. Retângulos representam as ações, retângulos vazios são as ações de persistência e vértices em cinza pontilhados representam os vínculos de exclusão mútua. Durante o primeiro instante de tempo, a única ação disponível de ser realizada (com todas as pré-condições atendidas) é *Comer(Bolo)*. No segundo instante de tempo, existem 4 literais que podem ser alcançados

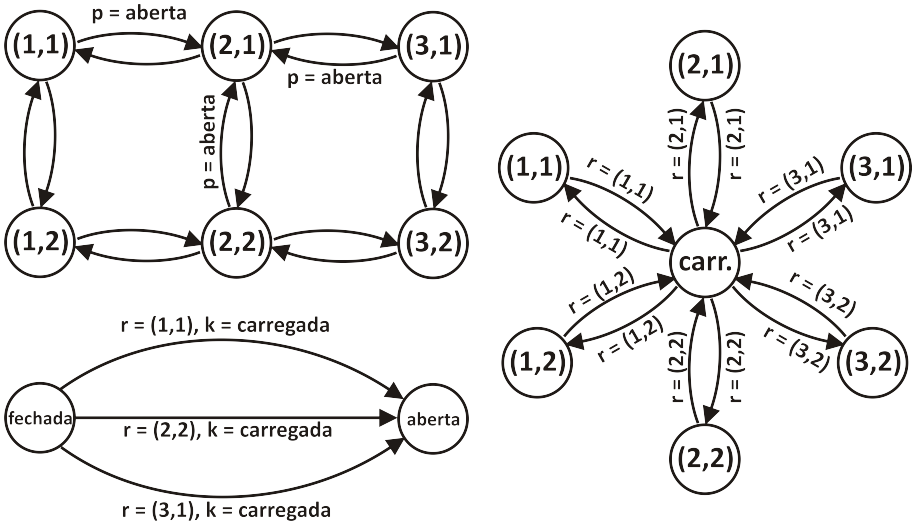

Figura 2.6: Exemplo: DTGs para o problema *Grid*

(*Ter(Bolo)*, *Comido(Bolo)* e suas negações). Apesar de todos serem acessíveis, literais interligados por vértices *mutex* não são acessíveis ao mesmo tempo, portanto não podem ser verdadeiras no mesmo nível.

O grafo de planejamento pode ser usado tanto como artifício para representar o problema e solucioná-lo, como para obter heurísticas. A partir do grafo de planejamento pode-se fazer seu relaxamento e extrair um grafo relaxado (RPG - *relaxed planning graph)*. Um grafo relaxado é um grafo de planejamento que omite os efeitos de ações que removem proposições, ou seja, todos os literais negativos produzidos por uma determinada ação são ignorados na geração do grafo. A partir da exclusão das ações de remoção, é possível estimar, sem a mesma precisão do grafo de planejamento completo, a complexidade do problema e extrair informações heurísticas que podem guiar a busca em um determinado problema [Hoffmann e Nebel 2001].

Outra forma de representar os problemas de planejamento é analisando as variáveis presentes no mesmo. Uma representação que pode ser utilizada é a representação do grafo causal (*Causal Graph* - CG) [Helmert 2006]. Um grafo causal é um grafo onde cada vértice lista as variáveis ou objetos presentes no problema, e esses vértices são conectados por arestas direcionadas, que indicam relacionamento direto entre eles.

Para gerar um grafo causal, é necessário inicialmente gerar um grafo de transição de domínio (*Domain Transition Graph* - DTG). O DTG é um grafo semelhante a um diagrama de estados, que apresenta para cada vértice um estado que um determinado objeto pode assumir, e cada aresta ligando dois vértices v_1 e v_2 indica que existe uma ação que faz o objeto analisado em questão modificar seu estado entre as ações. O exemplo presente na Figura 2.6 apresenta o DTG de um problema *Grid*, onde o objetivo é comandar um robô r entre posições demarcadas numa espécie de tabuleiro, e essas posições podem ter portas d entre elas. O robô deve então utilizar a chave k para abrir as portas.

No DTG superior esquerdo, cada vértice representa as posições que o robô r pode assumir. Algumas arestas exigem condições para serem executadas, no caso, que a porta d esteja aberta. No DTG inferior esquerdo, é apresentada a situação da porta d, que pode estar aberta ou fechada. No DTG direito, é apresentada a situação da chave k, com os estados que ela pode assumir, seja em uma das posições do *grid*, seja carregada pelo robô r [Helmert 2006].

A partir dos DTGs, é possível criar o grafo causal. Ao criar o grafo causal, usa-se uma variável para cada um dos DTGs desenvolvidos. Cada variável é representada por um vértice, e dois vértices, v_1 e v_2, que representam dois objetos, são conectados por uma aresta, se e somente se:

- O DTG de v_1 possui uma transição (aresta) que possui v_2 como condição. Nesse caso, v_1 depende de v_2, então cria-se uma aresta direcionada de v_2 a v_1, indicando a dependência;
- Em uma única ação, v_1 e v_2 são modificados. Nesse caso, ambas são modificadas em conjunto, portanto a aresta é bidirecional.

A Figura 2.7 apresenta o grafo causal para o problema *Grid*. São listadas 4 variáveis: r, que indica a posição do robô; a, que indica a situação do braço do robô (carregando ou não uma chave); d, que indica o estado da porta (aberta ou fechada); e k, que apresenta a localização da chave:

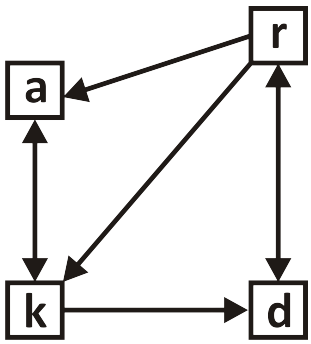

Figura 2.7: Exemplo: Grafo causal para o problema *Grid*

A partir de um grafo causal, é possível determinar a relação de dependência causal entre variáveis, e a partir dessa dependência é possível analisar as variáveis com maior importância dentre um problema de planejamento e quais as ações que podem gerar essas variáveis. Também é possível dividir o problema em subproblemas que envolvam apenas variáveis relacionadas entre si e solucioná-lo primeiro, ou determinar heurísticas a partir desse grafo.

Características

Analisando as formas de representação de um problema de planejamento, é possível extrair características que são inerentes ao problema em questão, e que podem auxiliar a definição de sua morfologia. A morfologia do problema pode auxiliar a extrair métricas como nós que serão expandidos, ou a decidir que tipo de algoritmo utilizar para resolvê-lo. Uma característica pode ser geral, presente em qualquer problema formalizado, ou específica, que informa dados relativos ao problema específico a ser trabalhado.

A extração de características pode ser realizada em várias representações, como a partir da PDDL de um problema. A seguir é apresentado o domínio Mundo dos Blocos.

```
(define (domain BLOCKS)
   (:requirements :strips)
   (:predicates (on ?x ?y) (ontable ?x) (clear ?x) (block ?x))
   (:action move
      :parameters(?b ?x ?y)
      :precondition (and (on ?b ?x) (clear ?b)
                    (clear ?y) (block ?b) (block ?y))
      :effect (and (on ?b ?y) (clear ?x)
              (not (on ?b ?x)) (not (clear ?y)))
   )
   (:action moveToTable
      :parameters(?b ?x)
      :precondition (and (on ?b ?x) (clear ?b) (block ?b) (block ?x))
      :effect (and (ontable ?b) (clear ?x) (not (on ?b ?x)))
   )
)
```

Observando detalhes dos domínios, é possível obter: o número de operadores e predicados de um problema, o número de pré-condições e efeitos em cada uma das ações, predicados estáticos (que não aparecem nos efeitos de nenhuma ação), números de classes, entre outros. A seguir tem-se um problema para o domínio proposto anteriormente.

```
(define (problem BLOCKS-1)
  (:domain BLOCKS)
  (:objects A B C)
  (:init (clear A) (clear B) (clear C) (ontable A)
         (ontable B) (ontable C) (block A) (block B) (block C))
  (:goal (and (on C B) (on B A) (ontable A))
)
```

Ao efetuar a análise sobre o arquivo de problema, é possível extrair o número de objetos do problema, fatos no estado inicial, fatos na meta, fatos estáticos e dinâmicos pertencentes a cada um deles. Combinando os dois arquivos PDDL, é possível extrair médias como: objetos por classe, ações por objeto, etc.

Nas representações em grafo também é possível extrair características. A partir do grafo DTG, é possível extrair quantas ações envolvem cada uma das variáveis do problema,

e a partir do grafo causal é possível definir ordens de dependência entre variáveis. Por último, no grafo de planejamento é possível extrair o número de fatos total e o número de ações por problema, e contabilizar médias de ações por objeto, ações que envolvam fatos do estado inicial e da meta, entre outros. A seguir, é citado um breve resumo de trabalhos que utilizaram abordagens semelhantes para extrair características de problemas.

Em [Vrakas et al. 2003], foram apresentadas 35 características que eram extraídas. As características foram classificadas de acordo com sua área de aplicação (tamanho do problema, complexidade, direcionalidade *forward* ou *backward*, e estimativas). A partir de sua extração, elas foram utilizadas para definir um conjunto de regras e posteriormente os parâmetros que seu planejador utilizaria, como direcionalidade da busca, heurística, relaxamento, entre outros.

[Roberts e Howe 2007] apresentou 32 métricas levantadas diretamente relacionadas a características apresentadas na linguagem PDDL e utilizou-as para representar diversos problemas de planejamento. A partir da representação gerada, foram treinados classificadores na tentativa de selecionar qual o melhor planejador resolveria o problema. O problema apresentado em [Roberts e Howe 2007] é chamado Problema da Seleção de Algoritmo e será abordado posteriormente na seção 2.3.

[Alhossaini e Beck 2013] também extraiu características de alguns problemas tentando definir o melhor planejador para resolver um determinado problema. Porém, diferentemente de [Roberts e Howe 2007], foram extraídas características específicas dos domínios, como quantidade de carros em problemas de transporte, ou quantidade de blocos no problema do mundo dos blocos.

2.2.2 Planejadores

O estudo sobre planejadores originou-se a partir de trabalhos que fazem uso de lógica de primeira ordem para criar solucionadores gerais de problemas nos mais diferentes domínios. Inicialmente, os planejadores eram utilizados para provadores de teoremas. Posteriormente, na década de 70, os planejadores começaram a atuar em problemas de aplicação prática do mundo real. A seguir será realizado um breve histórico a respeito dos planejadores. Serão apresentados planejadores clássicos e planejadores atuais, para exibir as técnicas que permitem que planejadores tenham resultados satisfatórios.

O planejador STRIPS [Fikes e Nilsson 1971] foi o planejador que forneceu novas noções e serve de inspiração para planejadores atuais. Nele, os estados do ambiente são considerados como um conjunto de predicados, as ações eram transformadores do ambiente. Foi proposta nele a ideia de decomposição da meta, e a busca era baseada em espaço de estados. Durante a busca, é necessário escolher um estado para expandi-lo e gerar novos estados sucessores. Para expandir um estado, são selecionadas todas as ações que possuem pré-condições atendidas naquele estado, e todas as ações são aplicadas sobre o

estado, onde seus efeitos podem adicionar e/ou remover predicados, criando assim novos estados.

O planejador *GRAPHPLAN* [Blum e Furst 1995] resolve seus problemas construindo um grafo de planejamento e efetuando a busca sobre esse grafo, considerando os predicados que devem pertencer à meta. Ele realiza a expansão do grafo de planejamento até que o nível se repita, e se nesse nível todos os predicados pertencentes à meta forem listados e não forem mutuamente exclusivos, seleciona as ações que criam os estados de forma sequencial.

O planejador *Heuristic Search Planner* (HSP) usa o algoritmo *Hill Climbing*, e aplica a função heurística proposta em [Bonnet e Geffner 1998], que gera uma estimativa do número de passos necessários para se chegar até a meta, a partir do estado corrente. A estimativa é realizada considerando a soma da quantidade de ações necessárias para se chegar em cada uma das partes da meta.

Baseado no *GRAPHPLAN* e no HSP, [Hoffmann e Nebel 2001] propõe o planejador *Fast Forward* (FF). O FF faz uma busca para frente no espaço de estados usando a função heurística, que estima a distância até a meta ignorando a lista de itens a serem deletados nos estados, que é uma forma de se relaxar o modelo para se chegar a solução com um menor custo. Outra diferença foi a proposta de uma nova estratégia de busca, que combina o algoritmo *Hill Climbing* com busca sistemática, para tentar escapar de máximos locais, o *Enforced Hill Climbing*. O FF usa também o conceito de ações úteis, que filtra ações mais promissoras a serem utilizadas e, em conjunto com o EHC, acelera o processo de busca. Mas caso o EHC venha a falhar, o planejador começa uma nova busca do zero, utilizando o algoritmo completo *Best First Search* guiado pela mesma heurística do EHC.

O FF serviu como base para o desenvolvimento de outros planejadores, como o *Fast Downward* (FD) [Helmert 2006], que computa os planos a partir de busca heurística no espaço de estados que podem ser visitados a partir do estado inicial. Porém, o FD usa uma função heurística diferente, chamada *causal graph heuristic* (heurística baseada em grafos causais), que não ignora os predicados excluídos.

O FD também propôs melhorias em seus algoritmos de busca: o *Greedy* BFS [Russell e Norvig 2009], que é uma versão modificada de BFS, com uma técnica chamada *deferred heuristic evaluation* (avaliação de heurística deferida) para mitigar a influência negativa de ramificações amplas. Também trata com *preferred operators* (operadores preferenciais), introduzidos pelo FD e que são similares as ações úteis do FF; o *Multi-heuristic* BFS: variação do *Greedy* BFS que avalia o espaço de busca usando múltiplas heurísticas, mantendo listas abertas separadas para cada heurística, e também fornecendo suporte aos *preferred operators*; e o *Focused iterative-broadening search*: um novo algoritmo de busca simples que não usa heurística, porém reduz o espaço vasto de possibilidades de busca focando num conjunto de operadores limitados derivados do GC.

O planejador *LAMA* [Richter e Westphal 2010] segue a mesma ideia dos outros sistemas citados acima usando busca heurística. Ele obteve a melhor performance entre todos os planejadores na trilha *satisficing* da IPC (*International Planning Competition*) em 2008. O *LAMA* usa uma variação da heurística do FF e a heurística derivada de *landmarks*, onde fórmulas proposicionais devem ser verdadeiras em todas as soluções de uma tarefa de planejamento. Dois algoritmos de busca heurística são implementados no *LAMA*: o *Greedy* BFS, com a tarefa de encontrar a solução o mais rápido possível, e o W-A*, que permite um balanceamento entre a rapidez e a qualidade de se encontrar uma solução. Ambos os algoritmos são variações de métodos usuais, usando listas abertas e listas fechadas.

[Xie et al. 2012] apresenta um método que usa *Greedy* BFS direcionado e uma combinação de *random walks* e avaliação direta de estados, que faz um equilíbrio entre exploração de estados não visitados e exploração de estados já avaliados. Essa técnica foi implementada no planejador *Arvand*, que competiu na IPC em 2011, o que melhorou a cobertura e a qualidade do planejador em relação a sua versão anterior. O algoritmo executa melhor que outros planejadores, especialmente em domínios onde muitos caminhos são gerados entre o estado inicial e a meta. Apesar disso, ele não possui boa performance em problemas que requerem busca exaustiva em grandes regiões do espaço de busca.

Em geral, para todos os planejadores apresentados é necessário usar uma combinação de técnicas de busca para garantir encontrar a solução. Algoritmos de rápida execução são selecionados inicialmente, e caso falhem, chamam algoritmos posteriores com garantia de completude. Essa abordagem pode ser ineficiente, pois muito tempo e esforço podem ser desperdiçados durante a primeira fase desses planejadores, caso o problema não seja resolvido.

2.2.3 Heurísticas

Em planejamento, uma heurística fornece uma estimativa de quantas ações, partindo de um estado qualquer, são necessárias para alcançar a meta. Neste caso, o caminho até a solução é composto por estados cujos valores heurísticos vão decrescendo a partir do estado inicial em direção à meta. Já estados fora desse caminho tendem a ter valores heurísticos piores (mais altos).

Uma heurística pode ser classificada em admissível e/ou consistente. Ela é considerada admissível se e somente se a estimativa de custo fornecida para um estado qualquer até a meta nunca supera seu custo real ($\forall\ s \in S, h(s) < g(s)$). Para uma heurística ser considerada consistente, para cada estado s, e seus sucessores s' gerados por uma ação a, o custo estimado para chegar na meta a partir de s não é maior que o custo da ação a mais o custo estimado para chegar à meta a partir de s' ($h(s) \leq h(s') + c(s, a, s')$) [Russell e Norvig 2009]. Uma heurística consistente sempre é admissível, mas a recíproca não é verdadeira.

Considerando as principais heurísticas em planejamento, é possível concluir que uma forma muito comum de se criar funções heurísticas em IA é simplesmente relaxar o problema, ou seja, imaginar o problema com menos restrições para encontrar uma solução relaxada que servirá como estimativa. Sendo um plano (sequência de ações) a solução para um problema de planejamento, uma proposta de heurística para planejamento seria desenvolver um modo de gerar planos relaxados, que na verdade seriam uma aproximação dos planos reais. Uma técnica bastante usada é a de ignorar a lista de itens a serem deletados de um estado ao se aplicar uma ação (*Ignoring Deletes*), o que pode gerar fatos contraditórios como: um bloco A estar sobre os blocos B e C ao mesmo tempo (considerando o exemplo do Mundo dos Blocos); mas isso não é algo que compromete a eficiência da busca, pois servirá apenas como estimativa durante a resolução do problema.

O planejador HSP introduziu a primeira heurística para planejamento, chamada heurística aditiva h_{add}. A heurística h_{add} não é admissível, mas sua estimativa é realizada de maneira rápida. Essa heurística é calculada analisando todos os fatos pertencentes à meta e comparando com os pertencentes ao estado a avaliar. Se um fato da meta não estiver presente no estado avaliado, é contabilizado o menor valor entre o custo de todas as ações que podem gerar aquele fato e a estimativa de seus predecessores. Se cada ação não possui custo explícito, se utiliza o custo uniforme (geralmente 1). Se o fato da meta estiver presente no estado avaliado, o valor contabilizado é 0. A heurística no final, como seu nome indica, é a adição dos custos de cada fato [Bonnet e Geffner 1998].

Para calcular a heurística de um estado s, usa-se a formulação $h_{add}(s) = h_{add}(G; s)$, onde G é o conjunto de fatos da meta. Para um conjunto de fatos F qualquer, é efetuada a soma do custo estimado de cada um dos seus fatos f: $h_{add}(F; s) = \sum f \in F\ [h_{add}(f; s)]$. Para um fato f qualquer, se esse fato pertence ao estado s, $h_{add}(f; s) = 0$. Caso contrário, é necessário analisar todas as ações a que geram o fato f. O custo real de aplicar a ação a é contabilizado e soma-se a ele o custo estimado dos predecessores de a: $h_{add}(Pre(a); s)$. Após contabilizar as estimativas de todas as ações, o menor valor é selecionado: $h_{add}(f; s) = min(a \in A(f))[cost(a) + h_{add}(Pre(a); s)]$.

Supondo o cálculo para o problema do mundo dos blocos resumido a seguir, calculando a heurística para o estado inicial:

- **Estado Inicial:** *Sobre(A, mesa)* $\wedge$ *Sobre(B, mesa)* $\wedge$ *Sobre(C, mesa)* $\wedge$ *Livre(A)* $\wedge$ *Livre(B)* $\wedge$ *Livre(C)* $\wedge$ *Livre(mesa).*
- **Estado Meta:** *Sobre(A, B)* $\wedge$ *Sobre(B, C)* $\wedge$ *Sobre(C, mesa).*
- **Mover(b, x, y):**
  - **pré-condições:** *Sobre(b, x)* $\wedge$ *Livre(b)* $\wedge$ *Livre(y)*
  - **efeitos:** *Sobre(b, y)* $\wedge$ *Livre(x)* $\wedge\neg$ *Sobre(b, x)* $\wedge\neg$ *Livre(y)*
- **MoverParaMesa(b, x):**

- **pré-condições:** *Sobre(b, x)* $\wedge$ *Livre(b)*
- **efeitos:** *Sobre(b, mesa)* $\wedge$ *Livre(x)* $\wedge\neg$ *Sobre(b, x)*

É possível analisar que os fatos *Sobre(A, B)* e *Sobre(B, C)* da meta não estão presentes no estado inicial, ao contrário do fato *Sobre(C, mesa)*, que já se encontra presente. Ao resolver o primeiro fato não presente (*Sobre(A, B)*), é preciso identificar quais ações podem gerar esse fato, e nesse caso, apenas a ação *Mover(A, mesa, B)* gera o fato descrito. Essa ação não tem custo explícito para sua execução, então é somado o custo uniforme 1. É necessário também contabilizar a estimativa para todos os seus predecessores, nesse caso, *Sobre(A, mesa)*, *Livre(A)* e *Livre(B)*. Como todos os predecessores já estão presentes no estado inicial, o valor somado é 0 para cada um dos valores. No final, o custo estimado para aplicar a ação é a soma de seu custo real (1) e de seus predecessores (0), totalizando 1. Se outras ações geram o fato analisado, seleciona-se o valor total menor entre todas as ações disponíveis. Ao contabilizar o segundo fato *Sobre(B, C)*, também existe apenas uma ação *Mover(B, mesa, C)* que tem custo 1 e com todos os predecessores *Sobre(B, mesa)*, *Livre(B)* e *Livre(C)* presentes no estado inicial, totalizando o custo estimado em 1. O fato *Sobre(C, mesa)* já está presente no estado inicial, então seu custo é 0. No fim, somando os 3 fatos, $h_{add}(s) = 2$.

A heurística h_{max} possui funcionamento semelhante à h_{add} [Bonet e Geffner 2001]. Analisa-se cada um dos fatos da meta comparando com o estado a ser avaliado. Porém, ao analisar todos os fatos, o fato com o maior valor estimado é selecionado, diferentemente da h_{add} onde soma-se os fatos. No exemplo acima, os 3 fatos avaliados (*Sobre(A, B)* e *Sobre(B, C)* e *Sobre(C, mesa)*)retornavam custos 1, 1 e 0 respectivamente. O maior valor é selecionado para se tornar a heurística, no caso $h_{max} = 1$. A heurística h_{max} é admissível, pois é baseada no número de ações necessárias para atingir a meta.

Outra heurística com funcionamento semelhante é a heurística *blind* ou cega h_{blind}. A heurística *blind* atribui 0 a todos os estados que cumprem a meta. Para os estados não-meta, todas as ações do problema são analisadas e o menor custo entre todas as ações é selecionado e atribuído a todos os estados [Helmert e Gabriele 2011].

A heurística h_{FF} [Hoffmann e Nebel 2001] introduzida pelo planejador FF, é baseada no conceito de relaxamento. Ela usa o sistema *GRAPHPLAN* para gerar um grafo de planos relaxados que ignora a lista de itens a serem deletados, e posteriormente esse RPG é usado para extrair a heurística.

Para calcular a heurística h_{FF} a partir do RPG, é necessário realizar uma busca regressiva no grafo relaxado da seguinte forma: Inicia-se pela ultima camada de fatos $F(n)$, que contém os fatos requeridos pela meta. Esses fatos que compõem a meta são agrupados em um subconjunto, chamado $G(n)$, dentro da camada de fatos $F(n)$. Para cada fato de $G(n)$, se ele está na camada de fatos anterior, ou seja, em $F(n-1)$, adicione um subconjunto $G(n-1)$ com esse fato dentro de $F(n-1)$, onde esse sub-conjunto

$G(n-1)$ representa fatos necessários para se chegar a meta e que estão na camada de fatos $F(n-1)$. Caso contrário, escolha uma ação de $A(n)$ que adiciona esse fato e adicione suas pré-condições em $G(n-1)$. O cálculo é interrompido quando atingir o conjunto $G(0)$, ou seja, ao subconjunto de fatos que levam a meta a partir da camada $F(0)$.

A Figura 2.8 mostra o RPG do problema de logística após ter passado pelo processo de criação dos subconjuntos $G(i)$. Em preto tem-se os fatos que compõem os subconjuntos $G(i)$ nas camadas de fatos $F(i)$, e em cinza as ações que são satisfeitas por esses fatos e que representam ações que se acredita serem necessárias para se chegar a meta.

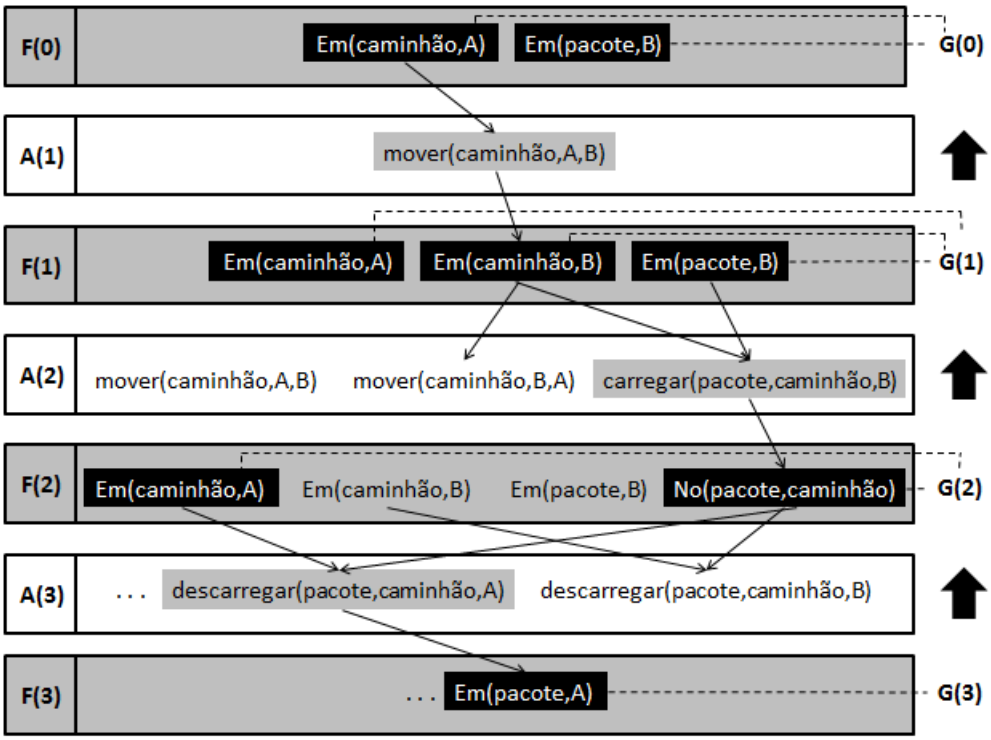

Figura 2.8: Exemplo: regressão no RPG

O custo de todas as ações $A(n)$ selecionadas durante o cálculo é o resultado da função heurística h_{FF}, ou seja, o numero de ações filtradas no RPG representa a estimativa de passos necessárias para se chegar a meta, a partir do estado inicial, como é ilustrado na Figura 2.9.

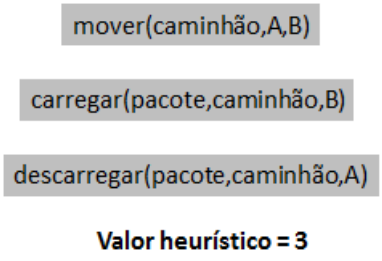

Figura 2.9: Exemplo: valor heurístico extraído do RPG

A heurística h_{CG} [Helmert 2006] é baseada no grafo causal, e calculada de modo

semelhante à geração do mesmo: supondo uma variável v, e dois estados s_1 e s_2 quaisquer que a variável pode assumir, a heurística h_{CG} contabiliza o custo necessário para converter a variável v do estado s_1 ao estado s_2.

Para calcular a heurística de um determinado estado do problema, analisa-se o custo para atender todas as variáveis na meta, comparando com as presentes no estado a calcular. Formalizando, a heurística para um estado é $h_{GC}(s') = \sum v \in V\ [cost(s(v), s_*(v))]$, onde V é o conjunto de todas as variáveis do problema, e v são todas as variáveis que devem assumir um estado $s_*(v)$ para atender a meta, porém atualmente estão na condição $s(v)$.

O cálculo do custo para transitar entre os estados $s(v)$ e $s_*(v)$ de uma variável v qualquer é feito da seguinte forma: analisa-se a relação de dependência no grafo causal. Se essa variável v não possuir dependência com nenhuma outra variável, o custo é contabilizado analisando as transições necessárias para mover de um estado para outro, em seu DTG. Caso contrário, é necessário contabilizar a soma dos custos para atender também a dependência da variável, somando ao custo final.

O cálculo de heurística é geralmente muito caro. Estima-se que o planejador FF gasta cerca de 80% de seu tempo calculando heurística durante a busca. Com isso, uma heurística deve ser bem informativa e podar o espaço de busca o suficiente para compensar seu cálculo. Mesmo considerando o alto custo, planejadores que utilizam busca heurística são os com maiores taxas de sucesso nos quesitos velocidade e qualidade da solução, como é possível analisar nos resultados das competições de planejamento IPCs, onde os campeões são planejadores guiados por heurística.

2.2.4 Portfólio de Planejamento

Um portfólio de algoritmos é uma estratégia para combinar vários algoritmos para resolver um determinado problema. A combinação de algoritmos é usada como uma caixa-preta, onde é necessária uma política para escolher a ordem que os algoritmos são executados [Vallati e Kitchin 2012]. O uso de portfólios de algoritmos na área de planejamento é recente, e foi impulsionado pela grande utilização de algoritmos de busca para solucionar problemas e pelo *No Free Lunch Theorem* (NFL), que afirma que entre dois algoritmos de busca A1 e A2, sempre que A1 solucionar melhor um grupo de problemas, o inverso deve ser verdade: A2 deve solucionar melhor outro grupo de problemas [Wolpert e Macready 1997].

Ao configurar um portfólio de planejamento, é necessário definir alguns parâmetros de configuração do portfólio: o escopo (independente ou específico de um domínio), alvo de otimização (tempo de execução, qualidade das soluções, maximizar quantidade de problemas resolvidos), quantidade de algoritmos, política de escalonamento dos algoritmos e as instâncias de treinamento. Um portfólio deve ter, pelo menos, melhor desempenho

que cada algoritmo individual que o compõe [Vallati e Kitchin 2012].

O planejador FDSS (*Fast Downward Stone Soup*) [Helmert e Gabriele 2011] possui como característica a combinação de planejadores de busca do tipo *forward* que incluem como principal componente o planejador *Fast Downward*. A característica principal do FDSS é a divisão compartilhada de tempo: supondo o fornecimento de 30 minutos para execução, ele limita todos os algoritmos para que executem com a mesma capacidade de tempo, nunca ultrapassando o tempo máximo permitido. O FDSS possui duas variações, uma com 15 e outra com 8 possíveis combinações de algoritmos.

Uma variação do FDSS, chamada convencionalmente de FDSS2 [Seipp et al. 2012] utiliza uma combinação de 21 algoritmos, e diversas variações de execução. Dentre as melhores combinações estão a divisão de tempo uniforme entre cada algoritmo, e a clusterização, onde ele organiza os algoritmos utilizando o *k-means* de acordo com uma pontuação pela resolução de problemas, onde cada problema é avaliado com um grau de dificuldade (quanto mais algoritmos solucionaram, mais fácil se torna o problema), e cada algoritmo recebe uma nota pelos problemas que resolveu. No final, os algoritmos selecionados são os com a maior pontuação. Ambos os planejadores FDSS e FDSS2 possuem duas linhas de desenvolvimento: a primeira, chamada OPT, se preocupa em encontrar planos com o menor custo possível; a segunda, chamada SAT, se preocupa em encontrar o plano no menor tempo possível.

O planejador PbP (*Portfolio-based Planner*) [Gerevini et al. 2009] possui funcionamento semelhante ao FDSS: uma série de combinações de planejadores é fornecida e cada um possui uma quantidade igual de tempo para ser executada. Porém, o PbP preza por utilizar apenas planejadores com a característica de resolver problemas usando *macro-actions*[4], que conseguem escolher diversas ações para serem executadas em paralelo, sem afetar os requisitos entre si ou o resultado final, se fossem executadas de forma sequencial.

Uma variação que melhora o PbP, adicionando outros planejadores, chamada PbP2 [Gerevini et al. 2011], adiciona também planejadores que em sua implementação principal não utilizavam *macro-actions*, criando versões que utilizam justamente pelo bom resultado apresentado na primeira versão do PbP.

O planejador *ArvandHerd* [Valenzano et al. 2012] usa recursos de multiprocessamento, que realiza execução de dois ou mais planejadores em paralelo. O *ArvandHerd* funciona basicamente com dois planejadores: *LAMA* e *Arvand*. O *LAMA* é o planejador campeão das últimas edições da competição de planejadores IPC, porém possui dificuldades com alguns problemas de planejamento. O planejador *Arvand* utiliza exploração pela técnica de Monte-Carlo, e a opção de utilizá-lo foi justamente pelo bom resultado obtido em problemas que o *LAMA* não conseguia resolver.

O *ArvandHerd* executa em paralelo uma instância do *LAMA*, que consome grande

[4]Grupo de ações de planejamento que podem ser executadas em paralelo sem afetar pré ou pós-condições entre si.

quantidade de memória, em um dos núcleos disponíveis. Para cada núcleo sobressalente, ele utiliza uma variação do *Arvand*, controlando a quantidade sucessores e passos a ser explorada pela técnica de Monte-Carlo, ocupando todos os núcleos e executando planejadores com características distintas de forma paralela.

2.3 Classificação

O uso de algoritmos de busca para resolver problemas de planejamento se mostrou eficiente em vários planejadores. Por muitos anos as pesquisas em IA focaram no desenvolvimento de novos algoritmos de busca para resolver problemas de planejamento. Apesar disso, nenhum algoritmo é melhor que todos os outros quando aplicados a problemas distintos [Wolpert e Macready 1997]. A partir dessas observações, uma das tendências nessa área é selecionar o melhor algoritmo para resolver um problema em particular [Kotthoff 2012], classificando os problemas e planejadores de acordo com sua afinidade e capacidade de resolução. Esse problema é conhecido na literatura como *Algorithm Selection Problem*, ou Problema da Seleção de Algoritmos [Rice 1976].

Os seguintes passos devem ser seguidos para se resolver o Problema da Seleção de Algoritmos [Rice 1976] e consequentemente da formação de um portfólio de planejadores: formular subclasses; determinar a existência do melhor mapeamento; determinar as propriedades que caracterizam a seleção; e computar métodos para resolvê-lo. Para problemas de planejamento, as subclasses podem ser planejadores, e a existência do melhor mapeamento pode ser provada depois de casos de teste aplicando cada planejador para resolver os problemas, tornando possível determinar o melhor planejador para cada problema. As propriedades podem ser as características extraídas das representações distintas que um problema pode assumir, e para resolver o problema computacionalmente, é possível usar métodos de classificação para determinar o melhor planejador para solucionar um determinado problema.

2.3.1 Classificadores

Ao se construir um classificador, seu treinamento será executado a partir de técnicas de aprendizado de máquina. Para construir um classificador, realizam-se três atividades: aquisição do corpus de documentos, que consiste na coleta de exemplos que identificam cada uma das classes a serem aprendidas; criação da representação dos documentos, extraindo características que conseguem identificar unicamente cada um dos exemplos; e indução do classificador, que usa um algoritmo de aprendizado para definir a partir de um conjunto de representações a qual categoria esse conjunto pertence, tornando possível a predição de classes para valores não utilizados durante o treinamento [de Souza Lima 2009].

O k-NN (*k - Nearest Neighbor*) [Aha e Kibler 1991] é um algoritmo de classificação que classifica cada nova instância a partir de exemplos de treinamento que estão mais próximos, ou seja, são considerados mais similares à instância a ser classificada. Ele funciona sempre analisando todos os outros exemplos classificados anteriormente e analisando com os k elementos mais próximos para definir sua classe. O valor k é um parâmetro do algoritmo, onde sua variação altera o comportamento do classificador. É possível usar qualquer fórmula de distância entre duas instâncias, como por exemplo, o cálculo de distância euclidiana em relação às características apresentadas.

A seguir, um problema com duas características foi apresentado em forma de um gráfico bidimensional, onde existem duas possíveis classes: *círculo* e *cruz*. Um novo indivíduo foi apresentado, representado pelo asterisco. A Figura 2.10 apresenta a classificação do k-NN, onde à esquerda o problema foi classificado usando $k = 1$ e à direita usando $k = 3$. É importante ressaltar que o valor de k altera o comportamento e para encontrar o valor ideal de k, é necessária uma fase de experimentação em cada problema distinto.

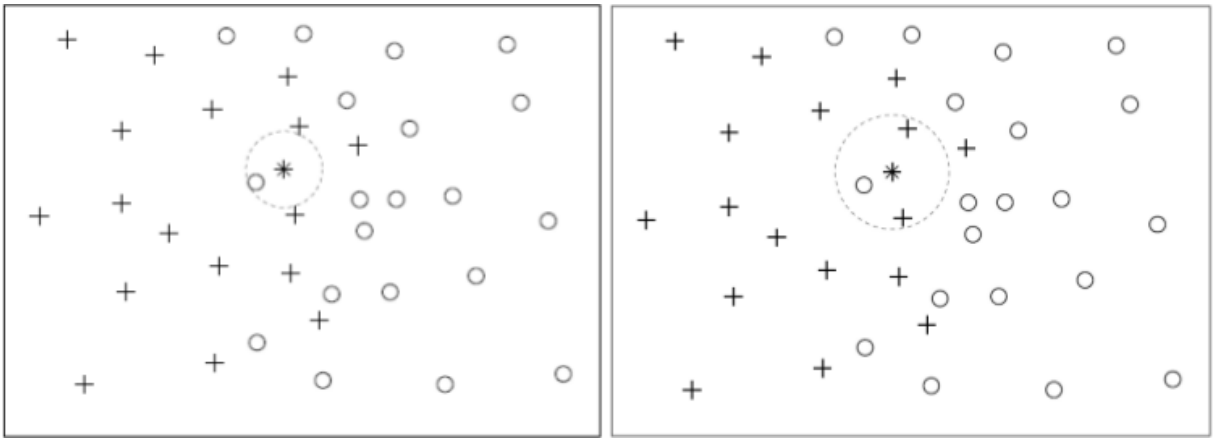

Figura 2.10: Classificação usando o k-NN

Na Figura 2.10, à esquerda o valor de k usado foi $k = 1$, onde compara-se o vizinho mais próximo pertence à classe *círculo*, tornando o indivíduo a ser classificado com também da classe *círculo*. À direita, foi usado $k = 3$, onde comparam-se os três vizinhos mais próximos. Dois pertencem à classe *cruz* e um a classe *círculo*. Como a classe *cruz* foi maioria, o novo indivíduo é classificado como pertencente à classe *cruz*.

Outra técnica usada para classificação é a árvore de decisão. A árvore de decisão é definida a partir de nós de teste e nós de decisão. Os nós de teste contêm uma pergunta, geralmente uma condição, sobre um dos atributos de um determinado teste. De acordo com a resposta da condição (verdadeira ou falsa), segue-se para um nó filho distinto. As folhas sempre são nós de decisão, que determinam a qual classe o documento testado pertence [Hunt et al. 1966].

Um dos principais algoritmos para geração de árvores de decisão é o algoritmo C4.5

[Quinlan 1993], implementado dentro do *framework* WEKA (*Waikato Environment for Knowledge Analysis*) [Witten e Frank 2005] com o nome J48. O algoritmo J48 usa os seguintes passos para ser treinado:

1. Escolher, a partir da raiz, um teste que maximize a separação das classes, onde cada valor de teste deverá ser associado a exemplos de classes diferentes;
2. Gerar um novo nó para cada valor do teste escolhido;
3. Se todos os exemplos que atingem o nó pertencem a uma mesma classe, então o novo nó é definido como uma folha associada a classe mais frequente dos exemplos do nó;
4. Caso contrário, retornar ao passo 1 para escolher um novo teste no nó e criar uma nova sub-árvore.

O algoritmo de Máquinas de Vetor Suporte (*Support Vector Machines* - SVM) [Burges 1998] é uma técnica de classificação que tenta encontrar uma equação capaz de definir um hiperplano ótimo capaz de classificar conjuntos linearmente separáveis. Após encontrar o hiperplano, basta analisar em que região um próximo problema se encontra para definir sua classificação. A técnica é apresentada no *framework* WEKA [Witten e Frank 2005] com o nome de SMO.

O Naïve Bayes (NB) é um classificador baseado no teorema de Bayes e é um algoritmo capaz de realizar o aprendizado indutivo com abordagem probabilística. A técnica permite computar a probabilidade de um novo problema pertencer a cada uma das classes, e então selecionar a classe com a maior probabilidade computada [Rish 2001].

Outro classificador que é possível utilizar é a rede de Kohonen, ou Kohonen-SOM (*Self Organizable Mapping*). A rede de Kohonen é uma rede neural com a habilidade de realizar mapeamentos que preservam a topologia entre os espaços de entrada e de saída. São utilizadas em muitos projetos industriais como ferramentas para resolver problemas práticos de difícil solução, com o objetivo de descobrir padrões significativos dos dados de entrada. Sua principal utilidade é na resolução de problemas não-lineares de alta dimensionalidade, como extração de características e classificação de imagens e padrões acústicos, controle adaptativo de robôs, equalização, modulação e transmissão de sinais [Kohonen et al. 2000].

Os neurônios em uma rede de Kohonen são colocados nos nós de uma grade (que apresenta uma topologia particular, que pode ser retangular, hexagonal etc.), que é usualmente de uma ou duas dimensões. Cada neurônio na grade é completamente conectado a todos os neurônios da camada de entrada. São apresentados os padrões de entrada à rede, e a cada padrão apresentado tem-se uma região de atividade na grade. A localização e natureza de uma determinada região variam de um padrão de entrada para outro. Assim sendo, todos os neurônios da rede devem ser expostos a um número suficiente de diferentes

padrões de entrada, garantindo assim que o processo de auto-organização ocorra de forma apropriada [Haykin 2001]. Um exemplo de rede de Kohonen é exibido na Figura 2.11.

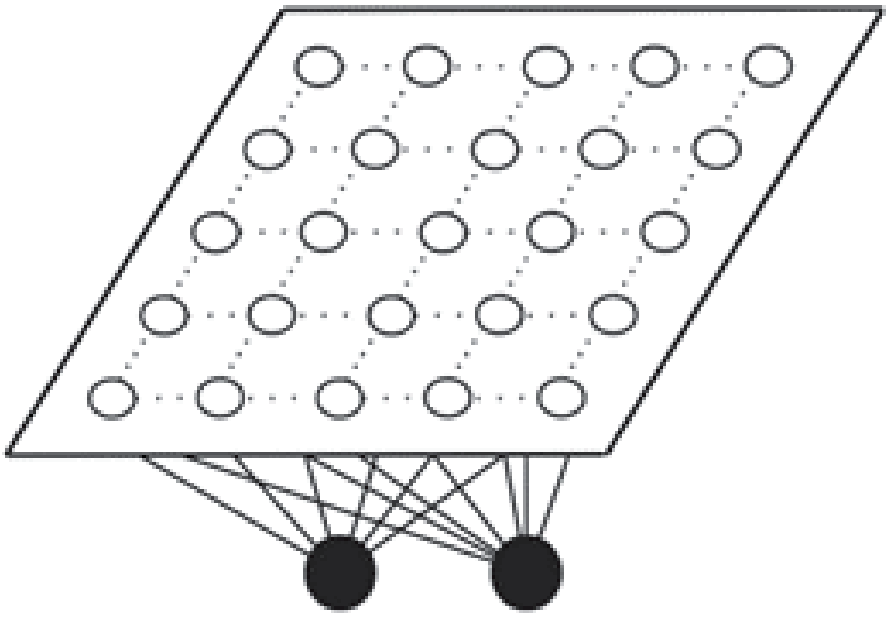

Figura 2.11: Topologias de uma rede de Kohonen

O processo de aprendizagem é baseado no aprendizado competitivo: as modificações dos pesos sinápticos são confinadas à vizinhança do neurônio ativado. A ordem global, ou seja, o equilíbrio da rede surge de interações locais. Os neurônios de saída competem entre si para serem ativados, de forma que apenas um neurônio de saída seja considerado vencedor. Os neurônios recebem estímulos (padrões de entrada) ao longo do processo competitivo de aprendizado. Serão considerados vencedores os neurônios que mais se assemelharem ao padrão de entrada, sendo que para esta comparação são utilizadas medidas de distâncias (normalmente utiliza-se a distância euclidiana).

2.3.2 Classificação em Planejamento

Fink [Fink et al. 1998] foi o precursor da utilização de classificação em planejamento. Ele comparou resultados de vários planejadores em um domínio específico e foi capaz de determinar dentre os algoritmos quais teriam os piores resultados para resolver esse problema, eliminando a possibilidade de usá-los.

O planejador *BUS* [Howe et al. 1999] foi um planejador de portfólio que utilizava técnicas de regressão linear para escalonar a ordem de execução dentre os diversos planejadores que ele armazenava, para resolver um determinado problema. Porém, em alguns casos testados, o tempo necessário para calcular o escalonamento era superior ao tempo de planejamento, tornando-o ineficiente até mesmo contra planejadores com estratégias simples.

Vrakas [Vrakas et al. 2003] propôs um planejador chamado HAP (*Highly Adjustable*

Planner - Planejador Altamente Ajustável) que analisava uma série de características extraídas de um determinado problema, e a partir delas foi capas de gerar regras usando aprendizado de máquina para determinar os melhores parâmetros que o planejador devia utilizar para resolver o problema. Em [Vrakas et al. 2003], foram apresentadas 35 características que eram extraídas. As características foram classificadas de acordo com sua área de aplicação (tamanho do problema, complexidade, direcionalidade *forward* ou *backward*, e estimativas). A partir de sua extração, elas foram utilizadas para definir um conjunto de regras e posteriormente os parâmetros que seu planejador utilizaria, como direcionalidade da busca, heurística, relaxamento, entre outros.

Em continuidade ao trabalho de Vrakas, Tsoumakas [Tsoumakas et al. 2004] propôs uma variação no planejador HAP chamada HAP-NN (*Highly Adjustable Planner Nearest Neighbor*), capaz de classificar os problemas usando suas características com o algoritmo K-NN, também com o objetivo de definir os melhores parâmetros para seu planejador.

Roberts [Roberts e Howe 2007] realizou um estudo profundo a respeito da classificação em planejamento: levantou dados de todas as competições e domínios existentes até a data de seu estudo, executou-os exaustivamente, apresentou 32 métricas levantadas diretamente relacionadas a características apresentadas na linguagem PDDL e utilizou-as para representar diversos problemas de planejamento, e a partir de características extraídas de propriedades da linguagem PDDL dos problemas, foi capaz de predizer a taxa de sucesso e o tempo de execução que um determinado planejador levaria para resolver o problema em questão. Apesar de acertar na predição de sucesso, o estudo não atingiu bons resultados ao predizer o tempo que o planejador levaria, dificultando o escalonamento dos planejadores em um portfólio.

Alhossaini [Alhossaini e Beck 2013] também trabalhou na predição de sucesso e na seleção do melhor planejador, porém trabalhou com características específicas dos domínios utilizados, como quantidade de carros em problemas de transporte, ou quantidade de blocos no problema do mundo dos blocos, o que dificultou na geração de um planejador de propósito geral.

Diferente dos trabalhos citados acima, Domshlak [Domshlak et al. 2010] propôs um algoritmo capaz de extrair as características e identificar qual a melhor heurística para resolver um determinado problema.

Cenamor [Cenamor et al. 2012] fez um trabalho semelhante ao de Roberts, e obteve melhores resultados ao estimar tempo e taxa de sucesso para seus problemas. Porém, as estimativas não foram muito efetivas em situações onde um domínio que não participou da etapa de treinamento era apresentado ao planejador.

2.4 Conclusão do Capítulo

Neste capítulo foram abordados conteúdos relacionados a planejadores apoiados por algoritmos de busca e técnicas de classificação em planejamento. Como foi possível observar, apesar de diversas propostas, não existe um planejador capaz de superar todos os outros ao resolver problemas distintos. Na tentativa de mitigar essa dificuldade, é proposto a seguir um sistema de planejamento combinando técnicas de aprendizado de máquina, que seja capaz de se adaptar a diversos problemas e solucioná-los, combinando planejadores distintos em um portfólio.

Capítulo 3

Metodologia

Neste capítulo são descritos os módulos de predição e o planejador de portfólio. Na Seção 3.1 serão apresentados detalhes da seleção dos componentes que juntos se transformam no planejador desenvolvido. São listados os algoritmos, heurísticas, características e problemas selecionados, o banco de dados e como os componentes são conectados para formar o planejador. Na Seção 3.2 são apresentados detalhes sobre o treinamento aplicado para os classificadores de forma *off-line*, detalhes sobre o levantamento de execuções e extração de características usados para classificar.

3.1 Arquitetura

Nesta seção serão descritos os componentes que compõe o planejador e sua arquitetura. A capacidade de analisar os problemas de planejamento e decidir qual o melhor planejador para resolvê-lo foi o objetivo principal do trabalho.

Durante a etapa do treinamento, os problemas foram executados em todos os planejadores fornecidos e seus resultados armazenados. Após isso, características de alguns problemas foram usadas juntamente com as estatísticas de execução levantadas para treinar os preditores. A etapa de treinamento foi realizada de forma *off-line*. Na etapa de planejamento, extraem-se as características de um problema para serem fornecidas ao preditor, onde o mesmo determina o melhor planejador para resolver o problema fornecido. A Figura 3.1 detalha a arquitetura do sistema proposto sintetizando a descrição apresentada anteriormente.

A seguir, são listados os planejadores, problemas, características e classificadores utilizados.

3.1.1 Planejadores

Um portfólio de planejamento exige uma seleção de planejadores que possam resolver com maior abrangência os problemas propostos, usando estratégias distintas, tornando-o

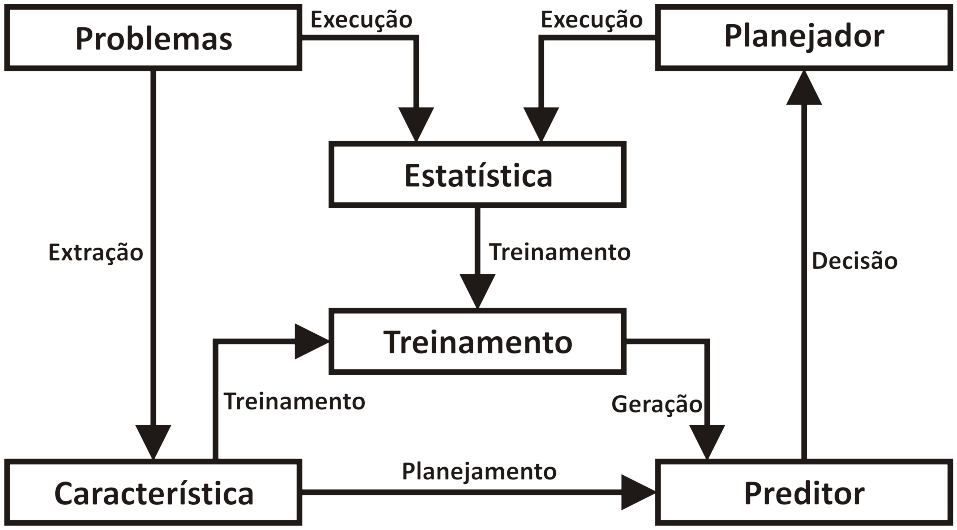

Figura 3.1: Arquitetura do sistema proposto

o mais completo possível. Durante a seleção dos planejadores optou-se por selecionar técnicas que obtiveram sucesso em outros planejadores já existentes, e que se completavam, permitindo abrangência maior sobre o número de problemas resolvidos. Foram selecionados os seguintes planejadores, sendo que cada combinação entre um algoritmo de busca e uma heurística é considerado um sistema de planejamento independente:

- **A* e sua variação *Weighted*-A*:** Utilizados no planejador *LAMA*, ambos os algoritmos aqui foram combinados com as heurísticas h_{CG}, h_{FF}, h_{add}, h_{max} e h_{blind}. O valor de w usado foi $w = 5$;
- ***Best-First Search* (BFS):** Usados pelos planejadores FF e FD, combinado com as heurísticas h_{CG}, h_{FF}, h_{add}, h_{max} e h_{blind};
- ***Enforced Hill-Climbing* (EHC):** Usado pelo planejador FF, combinado com as heurísticas h_{CG}, h_{FF}, h_{add}, h_{max} e h_{blind};
- ***LAMA-2011*:** Versão do planejador enviada para o IPC, utilizada sem variações;
- ***BJOLP*:** Versão do planejador enviada para o IPC, utilizada sem variações;
- ***LM-Cut*:** Usa uma heurística que define fatos que deverão ser verdadeiros ou, até mesmo, estáticos em algum momento do planejamento das ações. Então esses fatos são colocados em uma lista chamada *landmarks*, e todos os outros fatos considerados dinâmicos irão compor uma lista chamada *to-do*, ao longo do planejamento, cujo tamanho representa a heurística;
- **FF:** Versão do planejador enviada para o IPC, utilizada sem variações;
- ***Arvand*:** Versão do planejador enviada para o IPC, utilizada sem variações.

Portanto, em função do exposto anteriormente, foram considerados trinta e seis sistemas de planejamento. Todas as implementações de sistemas foram obtidas a partir do

site da IPC, onde excetuando o planejador *Arvand* [Xie et al. 2012], todos os algoritmos de busca que se tornaram sistemas de planejamento foram utilizados a partir da implementação do planejador *Fast Downward* [Helmert 2006], disponível em domínio público para utilização e testes.

3.1.2 Características

As características foram extraídas a partir de observações feitas em [Vrakas et al. 2003] e [Roberts e Howe 2007]. Foram selecionadas trinta e oito características que levantavam informações sobre o problema. Algumas das características que eram relacionadas à informações de requisitos da linguagem PDDL, usadas em [Roberts e Howe 2007] foram ignoradas. Informações baseadas em heurísticas utilizadas no HAP [Vrakas et al. 2003] também foram ignoradas durante a seleção. As características utilizadas são listadas na Tabela 3.1.

A Tabela 3.1 possui três colunas: na primeira é listado o código da característica, na segunda o detalhamento do nome e na terceira de qual representação (PDDL, Grafo de Transição de Domínio - DTG e Grafo de Planejamento - PG).

3.1.3 Problemas

Os domínios utilizados são aqueles que aparecem nas duas últimas competições da IPC (2008 e 2011). Especifica-se a seguir uma lista com os domínios selecionados e um rápido resumo do tipo de problema abordado em cada domínio:

- ***Barman*:** O objetivo do problema é controlar um robô barman, que é responsável por preparar drinks de acordo com as solicitações. As ações são manipular os recipientes e misturadores, e efetuar a entrega dos drinks aos clientes;
- ***Elevators*:** O objetivo do problema é gerenciar o transporte de pessoas em um hotel através dos elevadores. Os problemas apresentados se diferenciam com relação ao número de andares, número de pessoas e velocidade dos elevadores;
- ***Floortile*:** Tem como objetivo manipular robôs pintores de pisos. A meta envolve as cores que cada peça deve ter ao fim do problema e a restrição é que um robô nunca pode se posicionar sobre um azulejo recém-pintado;
- ***Openstacks*:** Neste problema, o objetivo é gerenciar o agendamento de pedidos em uma fábrica. É necessário atender um número de pedidos de diferentes produtos, porém com a limitação de produzir apenas um produto por vez;
- ***Parcprinter*:** No problema *parcprinter* tem-se que gerenciar uma impressora com diversas saídas de papel e capacidade de impressão simultânea em todas elas. O objetivo é escalonar as atividades de impressão para aproveitar a característica de cada saída (preto e branco, colorido, etc.);

Tabela 3.1: Características Levantadas

#Caract.	Nome	Origem
C01	Número de Fatos no Estado Inicial	PDDL
C02	Número de Fatos Estáticos	PG
C03	Número de Fatos Dinâmicos	PG
C04	Número de Fatos	PG
C05	Número de Fatos na Meta	PDDL
C06	Número de Ações	PDDL
C07	Porcentagem de fatos dinâmicos no estado inicial sobre todos os fatos dinâmicos	PG/PDDL
C08	Número de Fatos Dinâmicos no Estado Inicial	PG/PDDL
C09	Número de Objetos	PDDL
C10	Porcentagem de fatos estáticos	PG/PDDL
C11	Porcentagem de fatos da meta sobre todos os fatos dinâmicos	PG/PDDL
C12	Proporção entre fatos dinâmicos no estado inicial e fatos da meta	PDDL
C13	Número médio de ações por fatos dinâmicos	PG/PDDL
C14	Número de Predicados	PDDL
C15	Número médio de fatos por predicado	PG/PDDL
C16	Desvio padrão do número de fatos por predicado	PG/PDDL
C17	Número de Operadores	PDDL
C18	Número médio de ações por operador	PG/PDDL
C19	Desvio padrão do número de ações por operador	PG/PDDL
C20	Número de *Mutexes*	PG
C21	Número médio de *mutexes* por fato	PG/PDDL
C22	Desvio padrão do número de *mutexes* por fato	PG/PDDL
C23	Número médio de ações requisitando um fato	DTG
C24	Desvio padrão do número de ações requisitando um fato	DTG
C25	Número médio de ações adicionando um fato	DTG
C26	Desvio padrão do número de ações adicionando um fato	DTG
C27	Número médio de ações excluindo um fato	DTG
C28	Desvio padrão do número de ações excluindo um fato	DTG
C29	Proporção entre o número de ações adicionando um fato e ações o removendo	DTG
C30	Número médio de fatos por objeto	PDDL/PG
C31	Número médio de ações por objeto	DTG
C32	Número de Classes	PDDL
C33	Número médio de objetos por classe	PDDL
C34	Desvio padrão do número médio de objetos por classe	PDDL
C35	Número de ações com precondições presentes no estado inicial	PG
C36	Número de ações que geram fatos presentes na meta	PG
C37	Número de ações com precondições no estado inicial e que geram fatos da meta	PG
C38	Proporção entre o número de ações requerendo fatos do estado inicial e ações que adicionam fatos da meta	PG

- ***Parking*:** Neste problema deve-se controlar um chofer que deve estacionar diversos carros nas vagas disponíveis no estacionamento;
- ***Pegsol*:** Aqui deve-se solucionar o famoso jogo *peg solitaire*, onde existem diversas peças em um tabuleiro, e cada peça deve comer as outras com movimentos semelhantes ao jogo de damas. O objetivo é ter apenas uma peça no tabuleiro ao fim do jogo;
- ***Scanalyzer*:** O objetivo deste problema é manipular a logística de uma estufa, controlando um robô que deve andar entre os corredores e realizar atividades como regar, colher e plantar;
- ***Tidybot*:** Neste problema, o objetivo é organizar a casa. Uma série de robôs tem a capacidade de pegar objetos que estão fora do lugar e posicioná-los no lugar correto;
- ***Transport*:** No problema de transporte, deve-se transportar cargas entre cidades;
- ***Visit-all*:** Neste problema, deve-se visitar todas as cidades em um mapa, baseado no problema do caixeiro viajante, porém sem a restrição de não poder repetir a visita a uma cidade;
- ***Woodworking*:** O objetivo deste problema é gerenciar atividades de uma madeireira, onde é necessário envernizar, polir, cortar e realizar outras atividades com toras de madeira de acordo com as solicitações.

Para cada domínio listado, foram utilizados os vinte problemas da última competição como parâmetros de teste, totalizando 240 problemas utilizados ao fim dos testes.

3.1.4 Levantamento dos Dados de Execução

Geralmente ao se utilizar técnicas de classificação, o *benchmark* utilizado para treinamento se encontra disponível para acesso e utilização, como em tarefas de classificação de plantas. Porém, na área de planejamento, tal *benchmark* não existia. Desta forma, a primeira etapa a ser realizada foi a coleta de informações de execução.

O procedimento para solução do problema foi executado cinco vezes para cada sistema de planejamento citado na Seção 3.1.1 compondo o portfólio. Todas as informações são armazenadas, como maior e menor tempo, tempo médio, quantidade de execuções sem falha, custo médio. São as informações que serão utilizadas para treinamento dos classificadores, pois é possível definir nas cinco execuções qual o melhor planejador a utilizar para cada problema.

Todos os problemas foram executados com o limite de 30 minutos e 2GB de memória, num computador Intel Core i7 2.40GHz com 8 núcleos e 6GB de memória, utilizando apenas um núcleo para cada algoritmo, buscando recriar desta forma as condições de participação na IPC. O sistema operacional utilizado foi o Linux Mint 14 versão Nadia.

As informações são salvas num banco de dados, todas as informações de características e tempo de execução de cada problema. O diagrama de entidade-relacionamento (DER) que modela o BD utilizado pode ser visualizado na Figura 3.2:

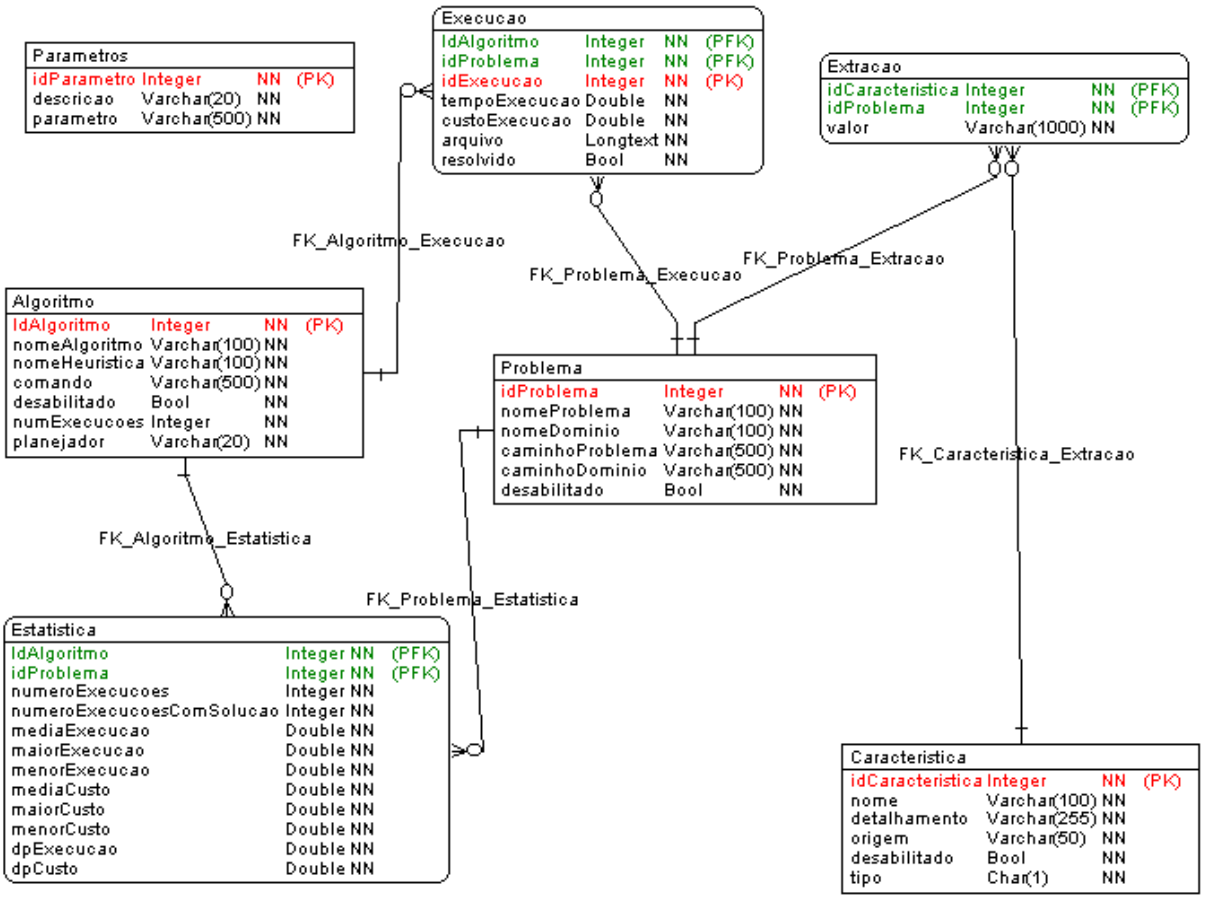

Figura 3.2: DER do banco de dados utilizado

As entidades Algoritmo, Característica, Problema e Parâmetros são auto-explicativas. Na entidade Execução, são armazenados os dados de uma execução de um algoritmo sobre um problema. Na entidade Estatística, agrupam-se todas as execuções de um algoritmo sobre um problema e levantam-se informações de problemas resolvidos, custo e tempo. Na entidade Extração são armazenados o valor de uma característica extraído de um determinado problema.

3.1.5 Classificadores

Os classificadores que foram utilizados pertencem ao *framework* WEKA [Witten e Frank 2005]. O WEKA é uma coleção de algoritmos de classificação e de pré-processamento de dados. Todo classificador construído no WEKA usa um arquivo padrão para entrada, treinamento e saída, bastando apenas fornecer esse arquivo para um classificador para

treiná-lo e fornecer um arquivo com instâncias de teste para receber as novas classificações. Então para usá-lo, a única preocupação é gerar o arquivo em formato ARFF. Nesse arquivo, é necessário atribuir algumas informações como:

- **Relação** (`@relation`): nome da série de dados;
- **Atributos** (`@attribute`): cada um dos atributos ou características representado. Cada atributo possui um nome e um tipo (*numeric*, *data* ou uma lista). É necessário determinar um atributo chamado ID que é a identificação única do problema (geralmente numérico);
- **Dados** (`@data`): a sequência de dados de cada um dos problemas. Cada dado possui o ID e um valor para cada um dos atributos listados acima.

A seguir, é apresentado um modelo de ARFF para o problema do tempo. São listados seis atributos: ID, que é a identificação de cada instância, *outlook* que representa a situação do tempo (*sunny* ou ensolarado, *overcast* ou nublado e *rainy* ou chuvoso); *temperature* que indica a temperatura, *humidity* que indica a umidade; *windy* que indica se está ou não ventando; e *play?* que indica se é possível ou não brincar fora de casa. Nesse problema, o objetivo é descobrir se posso ou não brincar na rua, ou seja, existem duas classes que cada uma das instâncias pode pertencer. Geralmente existem 2 arquivos ARFF: um que representa as instâncias de treinamento, usadas para treinar o classificador, e outro, que representa as instâncias de teste, para avaliar a taxa de sucesso do classificador gerado.

```
@relation weather

@attribute ID numeric
@attribute outlook { sunny, overcast, rainy }
@attribute temperature numeric
@attribute humidity numeric
@attribute windy { true, false }
@attribute play?  { yes, no }

@data

  1, sunny, 85, 85, false, no
  2, sunny, 80, 90, true, no
  3, overcast, 83, 86, false, yes
  4, rainy, 70, 96, false, yes
  5, rainy, 68, 80, false, yes
```

Para classificação em planejamento, os atributos são cada uma das características levantadas, e os dados são cada um dos problemas que serão classificados. Em nossa analogia, cada classe representa a utilização de um planejador específico, então, quando o classificador prediz que um problema pertence à classe LAMA, ele diz que o melhor

algoritmo usado para resolver o problema é o LAMA. Ao usarmos todas as trinta e oito características, teríamos quarenta atributos: o ID, cada uma das características e a classe, que é o melhor planejador para um determinado problema. Para testar a eficiência dos classificadores no problema, foram utilizados os algoritmos a seguir:

- **J48:** Implementação do algoritmo C4.5 com poda, sem poda e com redução;
- **Kohonen SOM:** Implementação da rede de Kohonen com variações na taxa de aprendizado (0.1 e 0.3);
- **Bayes:** Classificador usando a rede Bayesiana;
- **k-NN:** Variações com K entre 1, 3, 5 e 10;
- **SMO:** Implementação com padronização e normalização.

Com as variações, foram selecionados doze classificadores distintos ao fim das execuções. A opção pelos classificadores listados foi a utilização dos mesmos em outras abordagens de classificação em planejamento e pelo sucesso que demonstraram durante essas ocasiões.

3.2 Treinamento

Com os componentes reunidos, a etapa de treinamento consistiu em representar os problemas e classificá-los de acordo com seu melhor algoritmo. Nessa etapa foi necessário superar três dificuldades: como dividir o conjunto de problemas, como definir qual o melhor algoritmo que resolve o problema e como representar o problema para o treinamento.

Para dividir os conjuntos de problemas em treinamento e testes, duas abordagens foram utilizadas:

- **Aleatório:** Foram selecionados 60 dos 240 problemas aleatoriamente (25%), independentes de domínios, para teste. Os 180 restantes são os problemas de treinamento;
- ***Leave-a-domain-out*:** Baseada em [Cenamor et al. 2012], foram selecionados 3 domínios aleatoriamente dos 12 disponíveis (25%) para a etapa de testes. Os 9 domínios restantes são usados para treinamento.

A abordagem *leave-a-domain-out* se aproxima mais dos problemas que seriam abordados na IPC, pois não se tem conhecimento de quais os domínios seriam utilizados para a disputa. Com isso foram gerados nove casos de teste em cada abordagem, para serem executados para cada classificador.

Para definir qual o melhor algoritmo para resolver cada problema, utilizou-se as estatísticas de execução levantadas para classificação. Extraiu-se a média de cada algoritmo

em tempo e quantidade de problemas resolvidos e classificou-se em ordem crescente, permitindo identificar qual o melhor algoritmo para o problema. Em alguns problemas, mais de um algoritmo obteve os mesmos resultados de média e quantidade de problemas resolvidos. Nesses casos, considerou-se que se o classificador for capaz de selecionar qualquer um dos melhores planejadores, ele obteve sucesso.

Por último, selecionar as características corretas foi uma tarefa que exigiu maior análise. Inicialmente foi criada uma forma de representar facilmente essas características para identificá-las unicamente: Um vetor, com trinta e oito posições binárias (o mesmo número de características), onde se na primeira posição o valor fosse 0, a característica 1 não estava presente no teste, e se o valor fosse 1, a característica 1 era utilizada no teste. A Figura 3.3 apresenta um exemplo. No exemplo proposto na Figura 3.3, as características selecionadas foram 2, 3, 5, 8, 11, 14, 15, 17, 18, 20, 21, 23, 26, 32, 35 e 38.

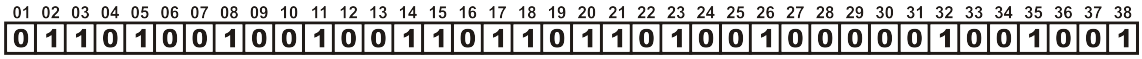

Figura 3.3: Vetor de características

Inicialmente foram selecionadas para treinamento as características com melhores resultados em [Vrakas et al. 2003] e também todas as características para treinamento. Foram notadas distinções dos resultados, definindo que a escolha das características está diretamente relacionada ao sucesso do preditor.

Como é de alto custo vasculhar todo o espaço de busca e testar todas as características, foi usado um algoritmo genético para selecionar a(s) melhor(es) combinação(ões) de característica(s) para resolver o problema. Cada indivíduo usa a mesma representação do vetor binário, e foram utilizados os seguintes parâmetros:

- **Número de gerações:** 100 gerações;
- **Número de indivíduos:** 100 indivíduos por geração;
- **Taxa de *crossover*:** 100%;
- **Taxa de mutação:** 0.5%;
- **Método de seleção:** Elitista, a cada geração seleciona-se os melhores indivíduos;
- **Método de mutação:** Um bit do vetor característica é aleatoriamente invertido;
- **Método de *crossover*:** Torneio simples com três indivíduos, onde os mesmos eram selecionados aleatoriamente e o com melhor *fitness* era o vencedor. Um detalhe a se analisar é que se um filho gerado fosse igual a qualquer outro indivíduo existente, o mesmo era submetido a uma mutação forçada para eliminar indivíduos repetidos.

Após a seleção de um vetor de características, seja no AG, seja manual, um arquivo ARFF padrão do WEKA é extraído com apenas as características determinadas pelo indivíduo, para seis conjuntos de problemas e utilizado para treinar os classificadores.

Como função de avaliação (*fitness*) do AG, para cada ARFF que identifica um problema gerado, doze preditores foram treinados, e o *fitness* se torna uma estrutura composta por três valores:

- A média da taxa de sucesso ao resolver todos os problemas dos nove conjuntos de testes, variando de 0 a 100, onde quanto maior melhor, procurando sempre a completude ao resolver os problemas;
- O consumo de tempo total gasto para os problemas resolvidos, onde quanto menor, melhor, na tentativa de agilizar o tempo para encontrar a solução;
- A pior taxa de sucesso de todos os preditores treinados para um determinado conjunto de testes, que realiza o desempate e busca equilibrar os preditores em uma alta taxa de sucesso, que varia de 0 a 100, onde quanto maior, melhor.

Após o treinamento, os preditores foram avaliados com todos os conjuntos de testes, para testar sua eficiência em relação aos problemas ainda não vistos. O método de treinamento e extração de características foi baseado em [Neto et al. 2014].

3.3 Composição do Portfólio

Com os preditores gerados, a composição do portfólio foi realizada da seguinte maneira: cada um dos n preditores com os melhores resultados seleciona um planejador, que segundo seu conhecimento é o melhor algoritmo para o problema em questão. Os planejadores selecionados são então escalonados uniformemente, onde cada um dos n planejadores selecionados possui $^{30}/_{n}$ minutos para ser executado, onde 30 minutos é o limite da IPC. Supondo que s classificadores selecionem o mesmo planejador, o tempo de execução padrão é multiplicado pelo número de vezes que ele foi selecionado, ou sejam $s \times {}^{30}/_{n}$. Uma abordagem semelhante foi proposta em [Seipp et al. 2012], onde foram criados k *clusters* com vários grupos de planejadores e o melhor algoritmo de cada um dos *clusters* foi selecionado recebendo $^{30}/_{k}$ minutos para ser executado. Detalhes sobre o resultado do portfólio são encontrados na Seção 4.4.

Capítulo 4

Experimentos e Discussões

Neste capítulo são apresentados os detalhes relacionados aos testes executados com o objetivo de comparar a performance dos planejadores usando classificação. Para tanto, este capítulo foi estruturado da seguinte forma: na Seção 4.1 são exibidos detalhes de como os testes foram realizados; na Seção 4.2 são apresentados detalhes sobre a execução do AG e seleção das características e na Seção 4.3 são detalhados os testes realizados para avaliar os classificadores e o planejador desenvolvido, além de discutir os resultados obtidos.

4.1 Metodologia

Os experimentos foram realizados em três etapas distintas: inicialmente executou-se o AG em busca do conjunto de características que resultaria nas melhores entradas para os classificadores. Posteriormente, executou-se uma comparação entre os classificadores gerados e os planejadores usado pelos classificadores, com base na afirmação que um portfólio deve sempre superar seus componentes individualmente para ser considerado eficiente. Nesta etapa, foram comparados a taxa de sucesso e o tempo gasto para gerar o plano de cada um dos planejadores nos dezoito casos de testes gerados. Após a comparação entre os componentes do portfólio criado, comparou-se os classificadores com os portfólios de planejadores com os melhores resultados nas últimas competições IPC, para demonstrar a eficiência dos planejadores gerados. Efetuou-se uma comparação relacionada ao tempo gasto dos classificadores para resolver os problemas e o tempo necessário para os portfólios de planejamento gerarem a solução. Por último, foi gerado o portfólio com os melhores classificadores gerados e comparou-se sua taxa de sucesso novamente com os melhores portfólios de planejadores da IPC.

4.2 Seleção de Características

Na etapa de seleção de características, o AG foi treinado com cem gerações. Porém, a partir da geração vinte não houve mudança significativa, ou seja, os melhores indivíduos foram encontrados antes do término da execução de todas as gerações. Ao concluir as cem gerações, dentre os cem indivíduos sobreviventes, os cinco indivíduos com melhor aproveitamento foram selecionados para realizar os testes nos classificadores. Cada indivíduo, representando um conjunto de características, recebeu um nome, de AG1 a AG5, respectivamente. Junto às características fornecidas pelo AG, também foram testadas as características definidas no planejador HAP, nomeada HAP e uma tentativa de usar todas as características para resolver o problema, nomeada ALL. O conjunto de características selecionado está descrito na Tabela 4.1 a seguir:

Tabela 4.1: Conjunto de Características Utilizadas

Vetor Característica	**Nome**	**Qtd. de caracts.**
00111011001001111111011101001010101000	AG1	21
11100011010001111111100010011100100001	AG2	20
11011101011010111100001011011000011001	AG3	21
10110010110011000011010101011010101111	AG4	21
00010000101001010011011111011001100010	AG5	17
00000000010100000010110000001110001111	HAP	12
11111111111111111111111111111111111111	ALL	38

Para cada conjunto de características, treinou-se todos os doze planejadores, executando os resultados de todos os dezoito conjuntos de testes. Executou-se também os resultados dos componentes usados pelos classificadores, e dos planejadores da última IPC: FDSS (OPT e SAT), FDSS2 (OPT e SAT), *Merge & Shrink* e *ArvandHerd* para efeitos de comparação.

4.3 Resultados

Nesta seção são apresentados os resultados encontrados. Serão apresentados os resultados dos componentes do portfólio, dos portfólios da IPC e dos classificadores gerados neste trabalho. Os resultados apresentados aqui são as médias de execução em cada um dos conjuntos de teste e a média geral para cada um dos planejadores.

A Tabela 4.2 exibe o resultado consolidado de cada um dos componentes. Nela, são apresentados o nome do planejador na primeira coluna, a média de sucesso em cada uma das abordagens nas duas colunas seguintes e a média de sucesso geral na coluna final da tabela. Os planejadores na Tabela 4.2 são apresentados em ordem crescente de sucesso geral. Os valores são apresentados em porcentagem.

Tabela 4.2: Média de sucesso dos componentes do portfólio

Planejador	**Média Aleatória**	**Média *leave-a-domain-out***	**Média Geral**
LAMA-2011	98.52%	98.52%	98.52%
W-A* (h_{add})	96.67%	97.41%	97.04%
A* (h_{add})	96.11%	96.30%	96.20%
A* (h_{FF})	94.44%	95.19%	94.81%
W-A* (h_{FF})	94.44%	93.70%	94.07%
BFS (h_{add})	91.11%	93.70%	92.41%
BFS (h_{FF})	91.11%	93.70%	92.41%
W-A* (h_{GC})	89.26%	91.48%	90.37%
A* (h_{GC})	92.22%	88.15%	90.19%
FF	85.93%	89.44%	87.69%
BFS (h_{GC})	84.63%	89.63%	87.13%
Arvand	84.44%	81.48%	82.96%
W-A* (h_{max})	78.15%	77.96%	78.06%
LM-Cut	78.89%	73.89%	76.39%
BFS (h_{max})	74.81%	77.04%	75.93%
A* (h_{max})	77.04%	71.30%	74.17%
BJOLP	70.19%	68.89%	69.54%
EHC (h_{add})	55.93%	74.81%	65.37%
EHC (h_{GC})	50.93%	72.59%	61.76%
W-A* (h_{blind})	61.67%	53.52%	57.59%
A* (h_{blind})	60.56%	52.59%	56.57%
EHC (h_{FF})	45.56%	66.48%	56.02%
EHC (h_{max})	43.33%	64.44%	53.89%
BFS (h_{blind})	53.15%	47.22%	50.19%
EHC (h_{blind})	20.37%	30.56%	25.46%

Na Tabela 4.2, é possível observar o domínio de alguns planejadores nos problemas selecionados. Na primeira posição geral obteve-se o *LAMA-2011*, campeão da IPC 2011. Em sequência, nas seis posições posteriores, aparecem todas as combinações dos algoritmos W-A*, A* e BFS com as heurísticas h_{add} e h_{FF}. Em sequência, os algoritmos A* e W-A* usando a heurística h_{CG} e o planejador FF, completando o ranking dos dez melhores planejadores.

Nas tabelas a seguir são exibidos os resultados para os classificadores gerados. Os resultados foram divididos em sete tabelas, uma para cada conjunto de características. Nessas tabelas, são apresentados o nome do classificador na primeira coluna, a média de sucesso em cada uma das abordagens, nas duas colunas seguintes e a média de sucesso geral na coluna final da tabela. Os classificadores são apresentados em ordem crescente de sucesso geral em todas as tabelas. Os valores são apresentados em porcentagem.

A Tabela 4.3 apresenta os resultados usando o conjunto de características ALL. Nela é possível notar que os três melhores classificadores são o SMO *Normalized*, J48 e J48

Unprunned. A taxa de sucesso, principalmente nos conjuntos de teste do grupo aleatório se mostrou positiva, porém não houve o mesmo resultado nos conjuntos *leave-a-domain-out*. A média de sucesso do SMO *Normalized* que é 96.76%, a melhor do conjunto usando todas as características, não foi capaz de superar o planejador *LAMA-2011*, que conseguiu média de 98.52%, o que mostra que o uso de todas as características não é interessante neste problema, onde foram extraídas trinta e oito características distintas. A quantidade de características provavelmente dificultou o treinamento dos classificadores.

Tabela 4.3: Média de sucesso dos classificadores, usando as características ALL

Classificador	**Média Aleatória**	**Média *leave-a-domain-out***	**Média Geral**
SMO *Normalized*	100.00%	93.52%	96.76%
J48	99.44%	93.15%	96.30%
J48 *Unprunned*	99.44%	93.15%	96.30%
k-NN (k = 10)	100.00%	92.41%	96.20%
k-NN (k = 5)	100.00%	92.41%	96.20%
k-NN (k = 3)	100.00%	90.37%	95.19%
SMO *Padronized*	100.00%	90.37%	95.19%
k-NN (k = 1)	98.89%	90.74%	94.81%
J48 *Reduced*	96.48%	89.63%	93.06%
SOM (LR = 0.1)	89.07%	86.67%	87.87%
Bayes	84.44%	90.56%	87.50%
SOM (LR = 0.3)	87.41%	85.37%	86.39%

Na Tabela 4.4 são apresentados os resultados dos classificadores usando a sugestão das características ótimas propostas em [Vrakas et al. 2003], chamada HAP. Os três melhores classificadores foram novamente algoritmos SMO e J48, porém agora em ordem e características de implementação distintas.

Os melhores resultados na Tabela 4.4 foram, respectivamente: SMO *Padronized*, J48 *Unprunned* e J48. Novamente, o melhor planejador, SMO *Padronized*, com média 96.39% não superou o resultado do *LAMA-2011* (98.52%). Esse fato pode ser explicado, pois o ano de desenvolvimento do HAP foi em 2003, e com isso o grupo de problemas da época provavelmente era mais simples que os problemas encontrados atualmente. Com isso, características que podiam ser efetivas para os problemas anteriormente não são interessantes para os problemas atuais, devido à simplicidade e a novos componentes que foram inseridos nas linguagens de representação de problemas de planejamento.

Na Tabela 4.5 são apresentados os resultados dos classificadores usando o conjunto de características encontrado no AG, chamado AG1. Entre os três melhores classificadores, novamente o algoritmo J48 dominou duas posições. Em ordem de classificação: J48 com média 99.35% e J48 *Unprunned*, com média 98.80%, ambos superiores à média do *LAMA-2011* (98.52%). Em terceiro, se localizou o k-NN ($k = 5, 10$), com média 97.13%. Com as características selecionadas nesse conjunto, foi possível superar os classificadores

Tabela 4.4: Média de sucesso dos classificadores, usando as características HAP

Classificador	**Média Aleatória**	**Média *leave-a-domain-out***	**Média Geral**
SMO *Padronized*	99.44%	93.33%	96.39%
J48 *Unprunned*	98.89%	90.19%	94.54%
J48	98.89%	89.44%	94.17%
J48 *Reduced*	99.26%	88.52%	93.89%
k-NN (k = 3)	100.00%	86.48%	93.24%
k-NN (k = 5)	100.00%	86.11%	93.06%
k-NN (k = 10)	99.26%	85.74%	92.50%
k-NN (k = 1)	98.89%	80.74%	89.81%
SOM (LR = 0.3)	87.78%	90.19%	88.98%
SMO *Normalized*	91.11%	86.48%	88.80%
Bayes	84.44%	90.56%	87.50%
SOM (LR = 0.1)	87.78%	85.74%	86.76%

individuais, o que tornou os classificadores J48 AG1 e J48 *Unprunned* AG1 prováveis candidatos a serem planejadores aptos a participarem em uma competição.

Tabela 4.5: Média de sucesso dos classificadores, usando as características AG1

Classificador	**Média Aleatória**	**Média *leave-a-domain-out***	**Média Geral**
J48	100.00%	98.70%	99.35%
J48 *Unprunned*	98.89%	98.70%	98.80%
k-NN (k = 10)	100.00%	94.26%	97.13%
k-NN (k = 5)	100.00%	94.26%	97.13%
k-NN (k = 3)	100.00%	93.33%	96.67%
k-NN (k = 1)	98.89%	94.26%	96.57%
SMO *Padronized*	100.00%	90.56%	95.28%
J48 *Reduced*	100.00%	90.19%	95.09%
SMO *Normalized*	96.48%	91.48%	93.98%
SOM (LR = 0.3)	92.59%	86.67%	89.63%
SOM (LR = 0.1)	91.11%	85.74%	88.43%
Bayes	84.44%	90.56%	87.50%

Na Tabela 4.6 são apresentados os resultados dos classificadores usando o conjunto de características AG2. Entre os três melhores classificadores, temos: k-NN ($k = 5, 10$), ambos com média 96.85%, e J48, com média 96.48%. Nenhum foi capaz de superar o algoritmo *LAMA-2011* (98.52%). Um detalhe que pode ter sido relevante é justamente as características selecionadas. O conjunto AG2 não apresenta a característica C04 - Número de Fatos, que foi usada em todos os outros conjuntos encontrados pelo AG. Não é possível afirmar que essa característica seja essencial, mas é possível afirmar que com mais fatos, consequentemente aumentam o número de estados, o que pode ser um fator determinante

ao escolher um planejador. É necessário um estudo mais detalhado para determinar qual o fator que não tornou esse conjunto tão eficiente quanto os anteriores.

Tabela 4.6: Média de sucesso dos classificadores, usando as características AG2

Classificador	**Média Aleatória**	**Média *leave-a-domain-out***	**Média Geral**
k-NN (k = 10)	100.00%	93.70%	96.85%
k-NN (k = 5)	100.00%	93.70%	96.85%
J48	98.15%	94.81%	96.48%
J48 *Unprunned*	98.15%	94.63%	96.39%
k-NN (k = 3)	100.00%	92.78%	96.39%
k-NN (k = 1)	98.89%	92.78%	95.83%
SMO *Padronized*	100.00%	90.74%	95.37%
SMO *Normalized*	96.48%	93.70%	95.09%
SOM (LR = 0.3)	97.59%	88.33%	92.96%
J48 *Reduced*	97.04%	87.41%	92.22%
SOM (LR = 0.1)	95.19%	86.11%	90.65%
Bayes	84.44%	90.56%	87.50%

A Tabela 4.7 apresenta os resultados dos classificadores usando o conjunto de características AG3.

Tabela 4.7: Média de sucesso dos classificadores, usando as características AG3

Classificador	**Média Aleatória**	**Média *leave-a-domain-out***	**Média Geral**
SMO *Padronized*	100.00%	97.78%	98.89%
SMO *Normalized*	100.00%	97.41%	98.70%
J48	99.44%	94.44%	96.94%
J48 *Reduced*	100.00%	93.70%	96.85%
J48 *Unprunned*	98.89%	94.81%	96.85%
k-NN (k = 3)	100.00%	92.22%	96.11%
k-NN (k = 5)	100.00%	92.22%	96.11%
k-NN (k = 10)	97.59%	93.70%	95.65%
k-NN (k = 1)	98.89%	91.85%	95.37%
SOM (LR = 0.3)	86.67%	89.44%	88.06%
Bayes	84.44%	90.56%	87.50%
SOM (LR = 0.1)	86.67%	84.44%	85.56%

Na Tabela 4.7, entre os três melhores classificadores, novamente o algoritmo J48 apareceu, e o algoritmo SMO voltou a figurar entre os três melhores. Em ordem de classificação: SMO *Padronized*, com 98.89% e SMO *Normalized*, com 98.70%, superando o *LAMA-2011* (98.52%). Em terceiro, o algoritmo J48, com 96.94%. Os classificadores SMO *Padronized* e *Normalized* AG3 são outros que conseguiram bons resultados e são possíveis planejadores finais.

A Tabela 4.8 apresenta os resultados dos classificadores usando o conjunto de características AG4. Os três melhores classificadores em ordem de classificação são: J48, com 97.50%, J48 *Unprunned*, com 97.22% e k-NN($k = 5, 10$), com 97.13%. Apesar de nenhum superar o *LAMA-2011* (98.52%), as médias foram superiores às encontradas no AG2.

Tabela 4.8: Média de sucesso dos classificadores, usando as características AG4

Classificador	**Média Aleatória**	**Média *leave-a-domain-out***	**Média Geral**
J48	97.59%	97.41%	97.50%
J48 *Unprunned*	97.59%	96.85%	97.22%
k-NN (k = 10)	100.00%	94.26%	97.13%
k-NN (k = 5)	100.00%	94.26%	97.13%
SMO *Padronized*	100.00%	93.33%	96.67%
k-NN (k = 3)	100.00%	93.33%	96.67%
k-NN (k = 1)	98.89%	94.26%	96.57%
J48 *Reduced*	98.70%	91.67%	95.19%
SMO *Normalized*	97.04%	92.04%	94.54%
SOM (LR = 0.1)	88.89%	86.48%	87.69%
Bayes	84.44%	90.56%	87.50%
SOM (LR = 0.3)	86.67%	86.67%	86.67%

A Tabela 4.9 apresenta os resultados dos classificadores usando o conjunto de características AG5.

Tabela 4.9: Média de sucesso dos classificadores, usando as características AG5

Classificador	**Média Aleatória**	**Média *leave-a-domain-out***	**Média Geral**
J48	100.00%	98.15%	99.07%
J48 *Unprunned*	98.89%	98.15%	98.52%
J48 *Reduced*	100.00%	95.00%	97.50%
k-NN (k = 3)	100.00%	92.22%	96.11%
k-NN (k = 1)	98.89%	92.22%	95.56%
k-NN (k = 10)	100.00%	90.93%	95.46%
k-NN (k = 5)	100.00%	90.74%	95.37%
SMO *Normalized*	97.59%	93.15%	95.37%
SMO *Padronized*	100.00%	88.70%	94.35%
SOM (LR = 0.3)	90.00%	88.33%	89.17%
Bayes	84.44%	90.56%	87.50%
SOM (LR = 0.1)	88.33%	83.70%	86.02%

Na Tabela 4.9, é possível notar que entre os três melhores classificadores em ordem de classificação são: J48, com 99.07%, J48 *Unprunned*, com 98.52% e J48 *Reduced*, com 97.50%. O J48 superou o *LAMA-2011* e o J48 *Unprunned* conseguiu o mesmo resultado

que o *LAMA-2011* (98.52%). Os classificadores J48 e J48 *Unprunned* AG5 conseguiram bons resultados e são possíveis planejadores finais.

Na Tabela 4.10 são apresentados os resultados consolidados com os dois melhores classificadores para cada um dos conjuntos de características e os três melhores planejadores testados, a fim de comparação de resultados.

Tabela 4.10: Resultados consolidados - classificadores e componentes do portfólio

Classificador / Planejador	**Característica**	**% Sucesso**
J48	AG1	99.35%
J48	AG5	99.07%
SMO *Padronized*	AG3	98.89%
J48 *Unprunned*	AG1	98.80%
SMO *Normalized*	AG3	98.70%
J48 *Unprunned*	AG5	98.52%
LAMA-2011	-	98.52%
J48	AG4	97.50%
J48 *Unprunned*	AG4	97.22%
W-A* (h_{add})	-	97.04%
k-NN (k = 10)	AG2	96.85%
k-NN (k = 5)	AG2	96.85%
SMO *Normalized*	ALL	96.76%
SMO *Padronized*	HAP	96.39%
J48	ALL	96.30%
A* (h_{add})	-	96.20%
J48 *Unprunned*	HAP	94.54%

Como é possível analisar, obteve-se seis planejadores treinados que foram capazes de superar todos os componentes, os tornando boas propostas para participação em competição ou uso prático. É possível analisar também que nenhum dos classificadores SOM ou Bayes conseguiram entrar na lista dos melhores, mostrando que eles não são tão eficientes nesse tipo de tarefa para os domínios de planejamento analisados. Por último, é importante ressaltar que os planejadores J48 são os mais efetivos no geral, estando presentes em oito dos quatorze melhores planejadores gerados a partir das características selecionadas. Uma outra observação é que os melhores planejadores usaram sempre características resultantes do AG, o que indica que usar todas as características pode confundir os preditores e as características sugeridas pelo HAP não são tão efetivas em problemas com maior nível de dificuldade.

Na Figura 4.1 são detalhados os tempos de execução em segundos dos seis melhores colocados na lista acima, que foram capazes de superar o *LAMA-2011*, e o próprio *LAMA-2011*, para fins de comparação. O tempo dos planejadores treinados neste trabalho apresentado nos gráficos inclui o tempo para extrair as características, executar o preditor e o planejador selecionado. A coluna horizontal apresenta o número do conjunto

de problemas resolvido e a coluna vertical apresenta o tempo em segundos necessário para se resolver o conjunto de problemas. Os problemas nomeados A1 a A9 compõe o grupo aleatório e os problemas L1 a L9 compõe o grupo *leave-a-domain-out*.

Na Figura 4.1, o planejador *LAMA-2011* é apresentado com maior destaque, na posição mais ao fundo, mostrando o tempo consumido em segundos para resolver cada um dos conjuntos de problemas. Em conjuntos que qualquer um dos planejadores não apresentou 100% de sucesso, seu marcador no gráfico foi zerado, ocasionando as quedas bruscas visuais no gráfico.

A partir da Figura 4.1 é possível analisar que nenhum dos planejadores de classificação superaram no total 1000 segundos para resolver os conjuntos. Já o planejador *LAMA-2011* manteve médias elevadas de tempo, todas superiores a 12000 segundos. É possível afirmar que, quando os planejadores preditores resolvem um problema, levam apenas 10% do tempo necessário que o *LAMA-2011* consome para resolver o mesmo. O planejador *LAMA-2011* resolveu com sucesso as instâncias A1, A3, A4, A6, A8 e A9 do conjunto aleatórios e L1, L3, L4, L6, L7 e L9, do conjunto *leave-a-domain-out*.

A partir da análise das tabelas e gráficos propostos, é possível dizer que os planejadores J48 AG1, J48 AG5, SMO *Padronized* AG3, J48 *Unprunned* AG1, SMO *Normalized* AG3 e J48 *Unprunned* AG5 foram capazes de superar todos os componentes de maneira individual, seja em taxa de sucesso, seja em tempo de execução. A bateria de testes a seguir busca analisar se os mesmos foram capazes de superar os melhores portfólios de planejadores da IPC 2011.

Inicialmente, a Tabela 4.11 exibe o resultado consolidado de cada um dos portfólios de planejamento. Nela, são apresentados o nome do planejador na primeira coluna, a média de sucesso em cada uma das abordagens nas duas colunas seguintes e a média de sucesso geral na coluna final da tabela. Os planejadores na Tabela 4.11 são apresentados em ordem crescente de sucesso geral. Os valores são apresentados em porcentagem. Nela, é possível observar que os melhores planejadores foram respectivamente, FDSS2 SAT, *ArvandHerd* e FDSS1 SAT.

Tabela 4.11: Média de sucesso dos portfólios de planejamento

Planejador	**Média Aleatória**	**Média *leave-a-domain-out***	**Média Geral**
FDSS2 SAT	98.33%	98.52%	98.43%
ArvandHerd	93.15%	95.19%	94.17%
FDSS1 SAT	92.22%	92.59%	92.41%
FDSS1 OPT	59.07%	55.00%	57.04%
FDSS2 OPT	58.52%	51.11%	54.81%
Merge & Shrink	18.52%	27.41%	22.96%

Na Tabela 4.12 são apresentados os resultados consolidados com os dois melhores classificadores para cada um dos conjuntos de características e os três melhores planejadores

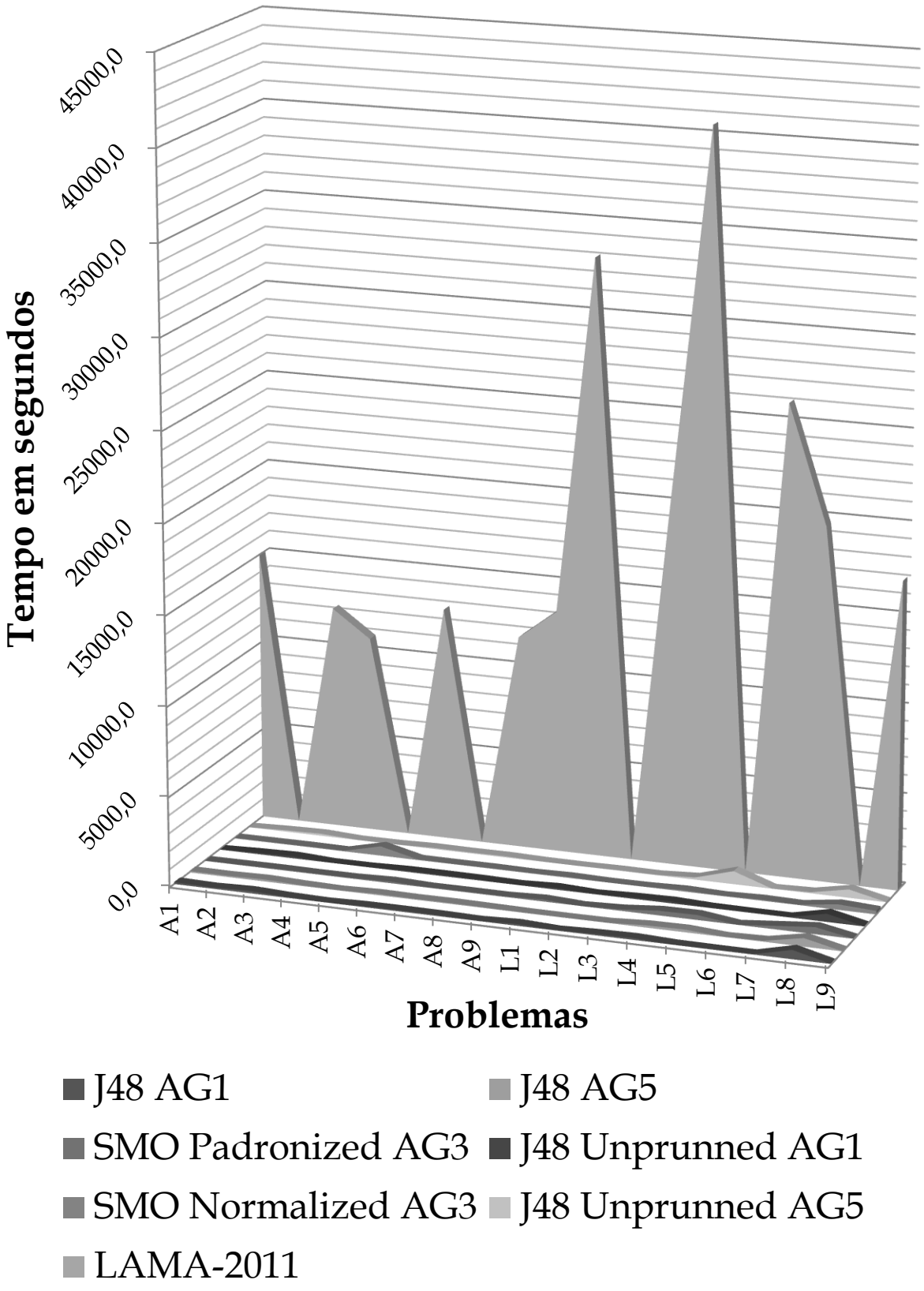

Figura 4.1: Tempo de execução - classificadores e componentes

testados, a fim de comparação de resultados.

Tabela 4.12: Resultados consolidados - classificadores e portfólios

Classificador / Planejador	**Característica**	**% Sucesso**
J48	AG1	99.35%
J48	AG5	99.07%
SMO *Padronized*	AG3	98.89%
J48 *Unprunned*	AG1	98.80%
SMO *Normalized*	AG3	98.70%
J48 *Unprunned*	AG5	98.52%
FDSS2 SAT	-	98.43%
J48	AG4	97.50%
J48 *Unprunned*	AG4	97.22%
k-NN (k = 10)	AG2	96.85%
k-NN (k = 5)	AG2	96.85%
SMO *Normalized*	ALL	96.76%
SMO *Padronized*	HAP	96.39%
J48	ALL	96.30%
J48 *Unprunned*	HAP	94.54%
ArvandHerd	-	94.17%
FDSS1 SAT	-	92.41%

Como é possível analisar, os seis melhores planejadores treinados foram capazes de superar todos os portfólios de planejadores da IPC 2011. Na Figura 4.2 são detalhados os tempos de execução em segundos dos seis melhores colocados na lista acima, que foram capazes de superar o FDSS2 SAT, e o próprio FDSS2 SAT, para fins de comparação.

Na Figura 4.2, o planejador FDSS2 SAT é apresentado com maior destaque, na posição mais ao fundo, mostrando o tempo consumido em segundos para resolver cada um dos conjuntos de problemas. Em conjuntos que qualquer um dos planejadores não apresentou 100% de sucesso, seu marcador no gráfico foi zerado, ocasionando as quedas bruscas visuais no gráfico. Na Figura 4.2, é possível analisar que nenhum dos planejadores de classificação superaram no total 500 segundos para resolver os conjuntos. Já o planejador FDSS2 SAT manteve médias superiores de tempo, acima de 700 segundos. Em alguns casos, o tempo gasto ultrapassou os 1000 segundos. Quando os planejadores preditores resolvem um problema, levam cerca de 80% do tempo necessário que o FDSS2 SAT consome para resolver o mesmo. O planejador FDSS2 SAT resolveu com sucesso apenas as instâncias A2, A3, A6, A7, A8 e A9 do conjunto aleatórios e L1, L3, L4, L6, L7, L8 e L9, do conjunto *leave-a-domain-out*.

A partir da análise apresentada, é possível dizer que os preditores J48 AG1, J48 AG5, SMO *Padronized* AG3, J48 *Unprunned* AG1, SMO *Normalized* AG3 e J48 *Unprunned* AG5 foram capazes de superar todos os portfólios de planejamento em taxa de sucesso e em tempo de execução.

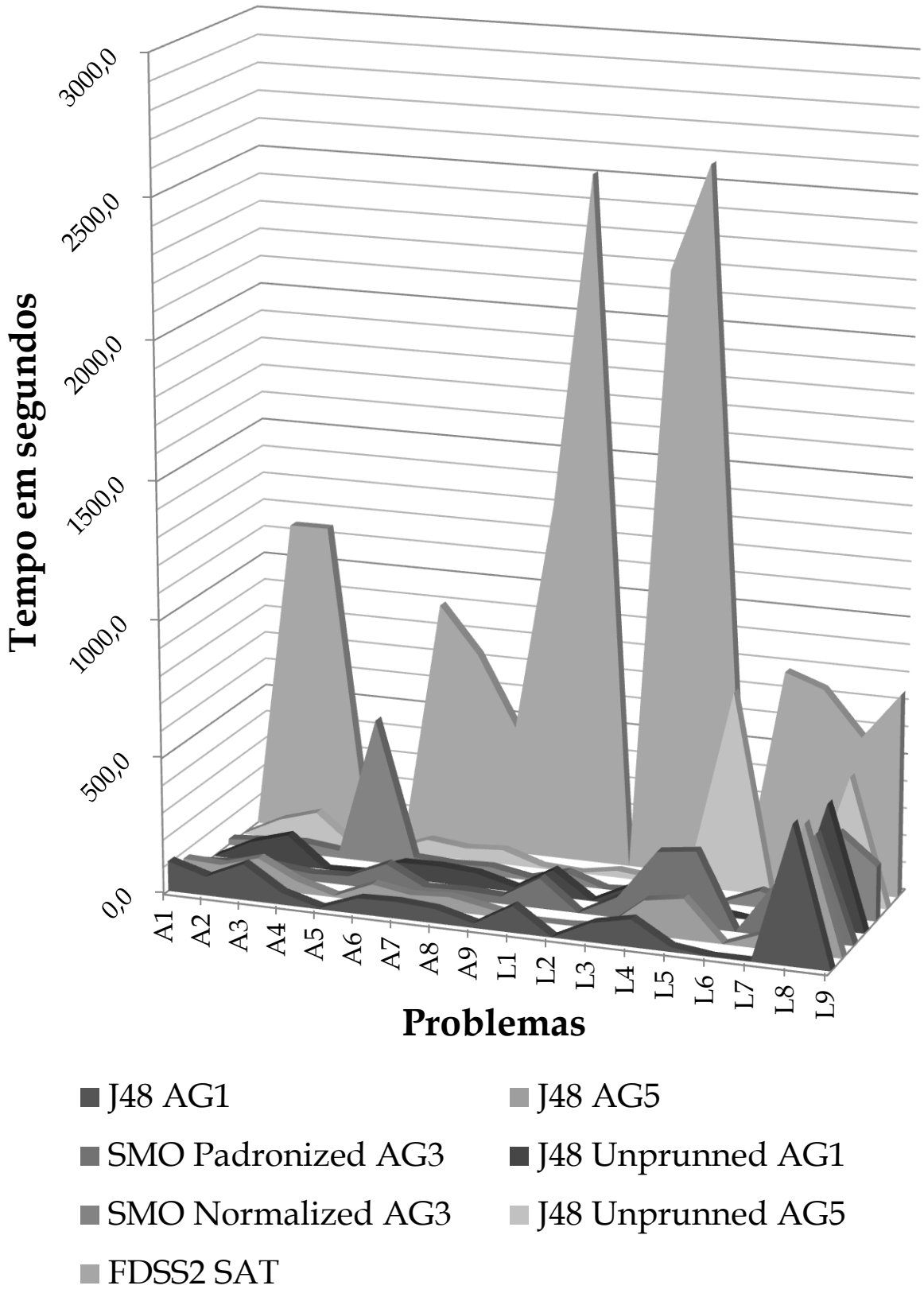

Figura 4.2: Tempo de execução - classificadores e portfólios

4.4 Portfólio Gerado

Como apresentado na Seção 4.3, seis preditores (J48 AG1, J48 AG5, SMO *Padronized* AG3, J48 *Unprunned* AG1, SMO *Normalized* AG3 e J48 *Unprunned* AG5) foram capazes de superar individualmente os planejadores com melhores resultados na última IPC. Porém, nenhum deles foi capaz de resolver 100% dos problemas propostos.

Na tentativa de melhorar a taxa de sucesso dos preditores gerados, foram combinados os seis melhores preditores em um portfólio, com comportamento descrito na Seção 3.3: cada um dos preditores selecionava um planejador, e cada um dos planejadores foi executado com a mesma quantidade de tempo (5 minutos para cada, totalizando os 30 minutos de limite da IPC). Se um planejador fosse selecionado por mais de um preditor, o mesmo tinha como tempo $s \times 5$ minutos para executar, onde s é o número de preditores que selecionou o planejador para execução.

O Gráfico 4.3 apresenta o comparativo entre o portfólio gerado e o melhor portfólio (FDSS2 SAT) da última IPC, em cada um dos 18 casos de testes.

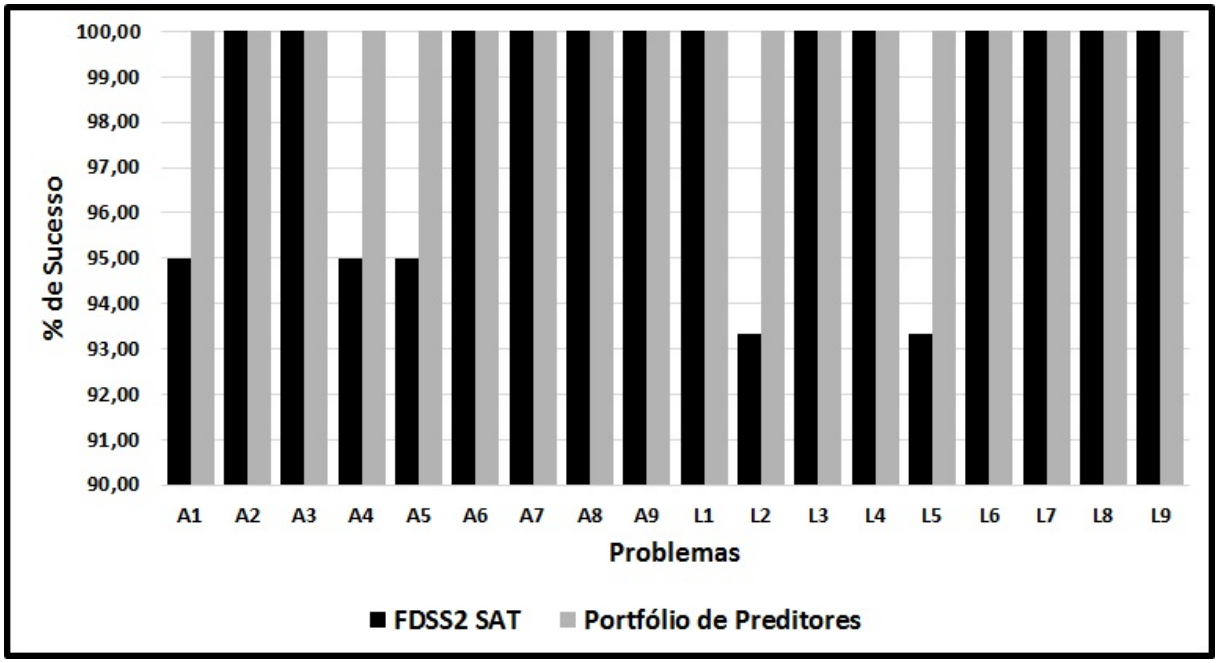

Figura 4.3: Resultados consolidados - Portfólio gerado e FDSS2 SAT

Como é possível analisar pelo Gráfico 4.3, o portfólio gerado foi sempre capaz de determinar, em todos os 18 casos de testes, um planejador que foi capaz de resolver 100% dos problemas. Isso mostra que os preditores, além de obterem bons resultados individualmente, são capazes de, se combinados, obterem resultados ainda melhores. No portfólio, os preditores foram capazes de contornar suas fraquezas e encontrar soluções em casos onde, individualmente, nem todos os preditores foram capazes de solucionar.

Capítulo 5

Aplicando Classificação em Planejamento Probabilístico

O planejamento probabilístico é uma extensão do planejamento clássico que trabalha sobre um ambiente não-determinístico, onde cada uma das ações possui uma probabilidade de ocorrer com sucesso ou não. Isso indica que, nem sempre quando executa-se uma determinada ação o estado sofre as mudanças dos fatos adicionados e excluídos pela ação.

Assim como no planejamento clássico, diversos planejadores foram propostos para resolver os problemas, incentivados principalmente pela *International Probabilistic Planning Competition* (IPPC) - Competição Internacional de Planejamento Probabilístico. O incentivo ao desenvolvimento de novos planejadores se deve principalmente pela mudança da linguagem utilizada para representar os problemas, de PPDDL (*Probabilistic Planning Domain Definition Language*) para RDDL (*Relational Dynamic Influence Diagram Language*), que permitiu a modelagem de ações concorrentes e de domínios com maior complexidade [Keller e Eyerich 2012].

Com o desenvolvimento de novos planejadores e observando os resultados individuais dos mesmos no IPPC, é possível notar que, apesar de dois planejadores se destacarem nos resultados finais (*PROST* [Keller e Eyerich 2012] e *Glutton* [Kolobov et al. 2012]), outros planejadores propostos conseguiram resultados positivos em problemas individuais, o que nos situa no mesmo problema enfrentado no planejamento clássico: nenhum planejador é capaz de superar totalmente os outros em todos os problemas [Vallati e Kitchin 2012].

Na área de planejamento probabilístico, não existe proposta semelhante à classificação dos problemas para predição dos melhores algoritmos. Baseado nessa observação, para comprovar que a classificação pode ser aplicada em outras áreas, foi proposto um classificador de planejadores probabilísticos, semelhante ao proposto para planejamento clássico, capaz de: extrair atributos de um problema informado e classificá-lo, determinando o planejador ótimo ou sub-ótimo que pode obter maior pontuação no problema proposto.

Este capítulo está organizado como descrito a seguir: A Seção 5.1 apresenta conceitos

sobre planejamento probabilístico e trabalhos correlatos. A Seção 5.2 mostra detalhes da arquitetura e do treinamento dos classificadores. Posteriormente, a Seção 5.3 apresenta os resultados encontrados e as conclusões de aplicação de classificação em planejamento probabilístico.

5.1 Conceitos

O planejamento probabilístico adiciona uma nova dificuldade ao planejamento: a incerteza que ao se solicitar a execução uma ação, a mesma tenha apenas uma porcentagem de chance de ocorrer com sucesso. Um problema de planejamento probabilístico pode ser solucionado em duas abordagens distintas: a primeira busca solucionar um problema e atender uma determinada meta; a segunda, utilizada pela IPPC, fornece uma função recompensa, que avalia após uma determinada quantidade de ações, chamada horizonte, em que situação o problema se encontra.

Na IPPC, o horizonte padrão é de quarenta ações. O funcionamento da competição é descrito a seguir: Usa-se um servidor que controla a execução das ações e o planejador, que efetua as decisões. Inicialmente, o estado inicial é fornecido pelo servidor ao planejador. O planejador então seleciona uma ação e envia ao servidor. O servidor então executa a ação (dentro de sua probabilidade), contabiliza a função recompensa e devolve o estado sucessor ao planejador. O ciclo continua até que o horizonte seja atingido. Cada ciclo em um determinado problema é chamado round, e cada problema geralmente possui trinta rounds em cada problema, e sua avaliação final é a média das recompensas de todos os rounds. Na IPPC o tempo consumido não é considerado na avaliação final. Porém, cada planejador possui apenas trinta minutos para tentar resolver um determinado problema.

Os ambientes clássico e probabilístico possuem características e objetivos distintos. No ambiente clássico, é interessante minimizar o tempo para gerar um plano. Já no ambiente probabilístico, é necessário maximizar a pontuação encontrada para um determinado problema.

Geralmente, problemas de planejamento probabilístico são solucionados com técnicas que resolvem MDPs (*Markov Decision Process* - Processos de Decisão Markovianos), devido a sua semelhança. Um problema de planejamento probabilístico, usando recompensas, é definido formalmente com o uma 5-upla [Rintanen 2006]:

- S: conjunto finito dos estados;
- I: distribuição probabilística sobre S (estado inicial);
- O: conjunto finito de ações ou funções parciais, que mapeiam cada estado como uma distribuição probabilística sobre S;
- C: $O \times S \rightarrow R$, que é uma função que converte as ações e estados em números reais, indicando o custo de uma determinada ação em um determinado estado;

- P: é uma partição de S em n classes $(C_1, ..., C_n)$ de estados observavelmente idênticos, onde $\cup(C_1, ..., C_n) = S$ e $\forall\ i, j, 1 \leq i < j \leq n, C_i \bigcap C_j = \emptyset$.

Os problemas de planejamento probabilístico geralmente são representados na linguagem RDDL. A RDDL [Sanner 2010] é uma linguagem uniforme onde os estados, ações e observações são representadas em variáveis parametrizadas, e a evolução do processo ou problema proposto é especificada via funções estocásticas aplicadas no estado corrente, permitindo concorrência entre as ações a serem executadas.

Um arquivo RDDL pode conter três tipos de seções principais: *domains*, onde detalhes sobre o domínio são apresentados; *non-fluents*, que apresentam a instanciação de não-fluentes, como estruturas fixadas entre os problemas; e *instances*: que apresenta a descrição do problema, o estado inicial, número de ações concorrentes, etc. As três seções serão detalhadas a seguir.

Inicialmente, é exibido um exemplo de arquivo com a seção *domains*:

```
domain <nome_domínio> {
   requirements = { <req_1> , ...  , <req_n> } ;
   types {
      <nome_tipo> :  <tipo_antecessor> ;
   }
   pvariables {
      <nome_var> :  { <tipo_var> , <tipo_dados> ,
                      default = <default> } ;
   }
   cpfs {
      <nome_cpf> ( <parâmetros> ) = <função> ;
   }
   reward = <função> ;
}
```

A seção *domains*, apresenta inicialmente o nome do domínio. Posteriormente, na seção *requirements*, são colocadas as exigências de linguagem que o *parser* RDDL deve possuir para interpretar o arquivo, como *reward-deterministic*, que indica que a pontuação do problema é baseada na função recompensa e *concurrent*, que permite que o domínio tenha concorrência entre as ações.

Na seção *types* são detalhados os tipos que as variáveis podem assumir. Na seção *pvariables* são listadas as variáveis, que podem assumir os tipos: *non-fluent*, que apresentam operadores que representam fatos estáticos; *state-fluent*, que são operadores que representam fatos dinâmicos; e *action-fluent*, que são operadores que controlam o comportamento das funções. Além disso, são apresentados para cada variável o tipo de dados (*integer*, *bool*, *real*, *object*, *enumerated*), e um valor padrão (*default*) para esse tipo.

Por último, são apresentadas as CPFs (*conditional probabilistic functions* - funções condicionais probabilísticas) que são as funções a serem aplicadas no problema e a função recompensa (*reward*).

A seguir, é detalhado um arquivo representando a seção *non-fluents*:

```
non-fluents <nome_nf> {
   domain = <nome_domínio> ;
   objects {
      <tipo_objeto> :  { <nome_objeto> } ;
   }
   non-fluents {
      // Aqui são listados os fatos estáticos.
   }
}
```

Na seção *non-fluents*, apresenta-se inicialmente seu nome e posteriormente a qual domínio ela pertence. São listadas duas subseções: *objects*, que lista todos os objetos que serão manipulados durante o problema e *non-fluents*, que apresenta a lista de fatos estáticos. A seção *non-fluents* é opcional, pois pode ser representada dentro do arquivo *instance*.

Por último, a seguir um arquivo representando a seção *instance*:

```
instance <nome_instância> {
   domain = <nome_domínio> ;
   non-fluent = <nome_nf> ;
   init-state {
     // Aqui são listados os fatos dinâmicos,
     // e também os estáticos, caso a seção
     // non-fluents não seja declarada.
   }
   max-nondef-actions = <valor> ;
   horizon = <valor> ;
   discount = <valor> ;
}
```

São apresentados inicialmente, nome da instância, nome do domínio e nome do *non-fluent*. Se o *non-fluent* não for especificado, pelo menos a seção *objects* deve ser inserida antes do *init-state*. Na *init-state* são apresentados os fatos dinâmicos e também os estáticos, se o *non-fluent* não for especificado. Por último, são listados três valores: *max-nondef-actions*, que determina o número de ações concorrentes no domínio; *horizon*, que limita o horizonte de ações para a IPPC; e *discount*, que pondera a média da função recompensa ao fim do round na IPPC.

5.2 Arquitetura

A arquitetura foi idêntica à proposta em planejamento clássico, com alterações nos componentes: problemas, características e planejadores. O processo de treinamento também foi modificado, pois não foi utilizado um AG. O conjunto de onze características selecionadas foi usado para treinamento de forma integral.

Para treinamento do planejador, foram levantados os resultados de cada um dos planejadores na IPPC 2011, avaliando individualmente cada um deles em um determinado problema. Após isso, características de alguns problemas foram usadas juntamente com as estatísticas de execução levantadas para treinar os preditores. A etapa de treinamento foi realizada de forma *off-line*. Na etapa de planejamento, extraíram-se as características de um problema e as mesmas foram fornecidas ao preditor para que ele decida o melhor planejador para resolver o problema fornecido.

Os planejadores selecionados foram os planejadores probabilísticos participantes da IPPC 2011:

- ***SPUDD*** [Hoey et al. 1999]: usa uma abstração dinâmica para resolver MDPs usando ADDs (*algebraic decision diagrams* - diagramas de decisão algébricas), que são generalizações dos BDDs (*binary decision diagrams* - diagramas de decisão binária);
- ***MIT-ACL*** [Ure et al. 2012]: usa uma regra de aprendizado baseada em valores para resolver as MDPs. A função-valor é representada usando uma aproximação linear;
- ***Beaver*** [Raghavan et al. 2011]: usa uma combinação do planejador SPUDD na parte de decisão e de UCTs (*Upper Confidence Trees*) para efetuar uma busca progressiva;
- ***Glutton*** [Kolobov et al. 2012]: planejador *anytime* capaz de solucionar MDPs usando o algoritmo RTDP. Ele consegue resolver o problema de forma ótima, desde que os recursos necessários (tempo e memória) sejam fornecidos;
- ***PROST*** [Keller e Eyerich 2012]: baseado no UCT, desenvolveu um procedimento que usa uma árvore semelhante às árvores de Monte-Carlo (*Monte-Carlo Tree* - MCT), onde a seleção das ações em cada instante é influenciada pelas decisões anteriores.

As características foram extraídas a partir de observações feitas sobre os arquivos RDDL que compunham o problema e no manual que descreve detalhadamente a linguagem [Sanner 2010]. Foram selecionadas onze características que detalham informações sobre fatos, concorrência, medidas de desempenho e representação de estados de cada um dos problemas. As características utilizadas são listadas na Tabela 5.1 a seguir:

Os domínios utilizados foram os presentes na IPPC 2011. Especifica-se a seguir uma lista com os domínios selecionados e um rápido resumo do tipo de problema abordado em cada domínio.

- ***Crossing Traffic***: Neste problema, um robô é posicionado em um *grid*, e deve chegar na meta desviando de diversos obstáculos que aparecem aleatoriamente. Se

Tabela 5.1: Características Levantadas

#Caract.	Nome
C01	Número de Fatos no Estado Inicial
C02	Número de Variáveis *non-fluent*
C03	Número de Variáveis *state-fluent*
C04	Número de Variáveis *action-fluent*
C05	Número Total de Variáveis
C06	Número de Fatos *non-fluent*
C07	Número de Objetos
C08	Número de CPFs
C09	Número de ações concorrentes
C10	Número de classes
C11	Número de objetos na função recompensa

o obstáculo atinge o robô, esse não pode mais se mover. O objetivo é chegar na meta com o mínimo de passos possível;

- ***Elevators***: No problema *elevators*, um número de elevadores é fornecido e deve coletar todos os passageiros e transportá-los para seu destino. Os locais de coleta de passageiros e destinos possíveis são apenas o primeiro e último andar de um prédio. O objetivo é levar os passageiros aos seus destinos sempre que um novo passageiro surgir;
- ***Game of Life***: Neste problema, é representado o jogo da vida ou *game of life*, um problema conhecido na área de autômatos celulares. O problema é apresentado em forma de um *grid*, e as regras são simples: uma célula morre se tiver menos que duas ou mais que células vivas em seu lado, uma célula sobrevive se tiver duas ou três células vivas ao seu lado, e uma célula morta revive se tiver exatamente três células vivas ao seu lado. Seu objetivo é manter o maior número de células vivas ao final do horizonte;
- ***Navigation***: O problema *navigation* é semelhante ao *crossing traffic*: um robô disposto num tabuleiro com o objetivo de chegar até a meta. Porém, cada célula tem uma probabilidade que o robô desapareça ao pisar nela;
- ***Reconnaissance***: Neste problema, é necessário controlar um robô em um *grid*. Estão dispostos pelo *grid* a base, células contaminadas e objetos espalhados. O robô possui três ferramentas: uma para detectar água, outra para detectar seres vivos e outra para tirar fotos. Alguns objetos podem estar contaminados, e ao analisar objetos contaminados com uma ferramenta, a mesma também é contaminada. Para descontaminar a ferramenta, é necessário ir até a base. O objetivo é encontrar o maior número de seres vivos e fotografá-los;
- ***Skill Teaching***: No problema *skill teaching*, controla-se um agente que tenta ensinar uma série de habilidades a um estudante, usando aulas e questões. Cada

estudante tem um nível de conhecimento em cada uma das habilidades, indicando sua probabilidade de acerto. O objetivo é aumentar esse nível de conhecimento e resolver o máximo de questões;

- ***SysAdmin***: Neste problema, controla-se o administrador do sistema que é responsável por manter uma rede de computadores ativa. Essa rede pode assumir diversas topologias e conter diversos computadores. Cada computador pode estar ou não em funcionamento. O objetivo é manter o máximo de computadores ativos na rede ao fim do problema;
- ***Traffic***: Aqui, deve-se gerenciar um agente que controla o tráfego de veículos em cruzamentos de uma cidade. O objetivo é justamente diminuir o engarrafamento de carros em determinadas vias, desafogando ou desviando o trânsito para vias alternativas.

Para cada domínio listado, foram utilizados os dez problemas da última competição como parâmetros de teste, totalizando oitenta problemas utilizados. Para cada problema, foi selecionada a pontuação de cada um dos cinco planejadores listados. A partir desses dados, foi possível definir qual o melhor planejador deve-se utilizar para resolver cada um dos problemas listados.

Com os componentes reunidos, a etapa de treinamento consistiu em representar os problemas e classificá-los de acordo com seu melhor algoritmo. Nessa etapa foi necessário superar três dificuldades: como dividir o conjunto de problemas, como definir qual o melhor algoritmo que resolve o problema e como representar o problema para o treinamento.

Para dividir os conjuntos de problemas em treinamento e testes, dois conjuntos de problemas foram utilizados: um deles, selecionando sete problemas de cada um dos domínios de forma aleatória para treinamento; e em outro, selecionando cinco domínios, totalizando cinquenta problemas, e usando para treinamento (*leave-a-domain-out*).

Para definir qual o melhor algoritmo resolvia cada problema, foram utilizadas as estatísticas de execução levantadas para classificação. Extraiu-se a pontuação de cada planejador e classificou-se em ordem crescente, permitindo identificar qual o melhor planejador para o problema. As estatísticas de execução definiram as classes para treinamento dos classificadores, e cada problema foi representado com o conjunto de onze características listadas, extraídos em arquivos ARFF seguindo o padrão usado pelo WEKA.

5.3 Resultados e Discussões

Nesta seção são apresentados os resultados encontrados ao treinar os doze classificadores distintos com dois conjuntos de problemas. Inicialmente, são exibidos os resultados dos planejadores probabilísticos na competição. Posteriormente, são apresentados os resultados obtidos pelos preditores em cada um dos conjuntos de problemas. A Tabela 5.2

apresenta os resultados dos planejadores no IPPC 2011, em ordem de classificação. É importante ressaltar que os números apresentados são a média da função resultado para cada um dos oitenta problemas distintos, e quanto maior essa média, melhor é seu resultado. Na Tabela 5.2, os planejadores *PROST* e *Glutton* obtiveram pontuação muito superior aos outros competidores. Apesar disso, em alguns problemas específicos os outros planejadores obtiveram melhores resultados, reforçando o fato que não é possível criar um planejador que seja superior em todos os tipos de problemas.

Tabela 5.2: Resultados IPPC 2011

Planejador	**Pontuação**
PROST	0.874
Glutton	0.795
SPUDD	0.297
Beaver	0.245
MIT-ACL	0.107

A Figura 5.1 apresenta a pontuação dos 24 planejadores treinados a partir de preditores. Essa pontuação indica a média da pontuação dos planejadores selecionados pelos preditores em cada um dos 80 problemas descritos na IPPC. Os planejadores *PROST* e *Glutton* também são apresentados para fins de comparação.

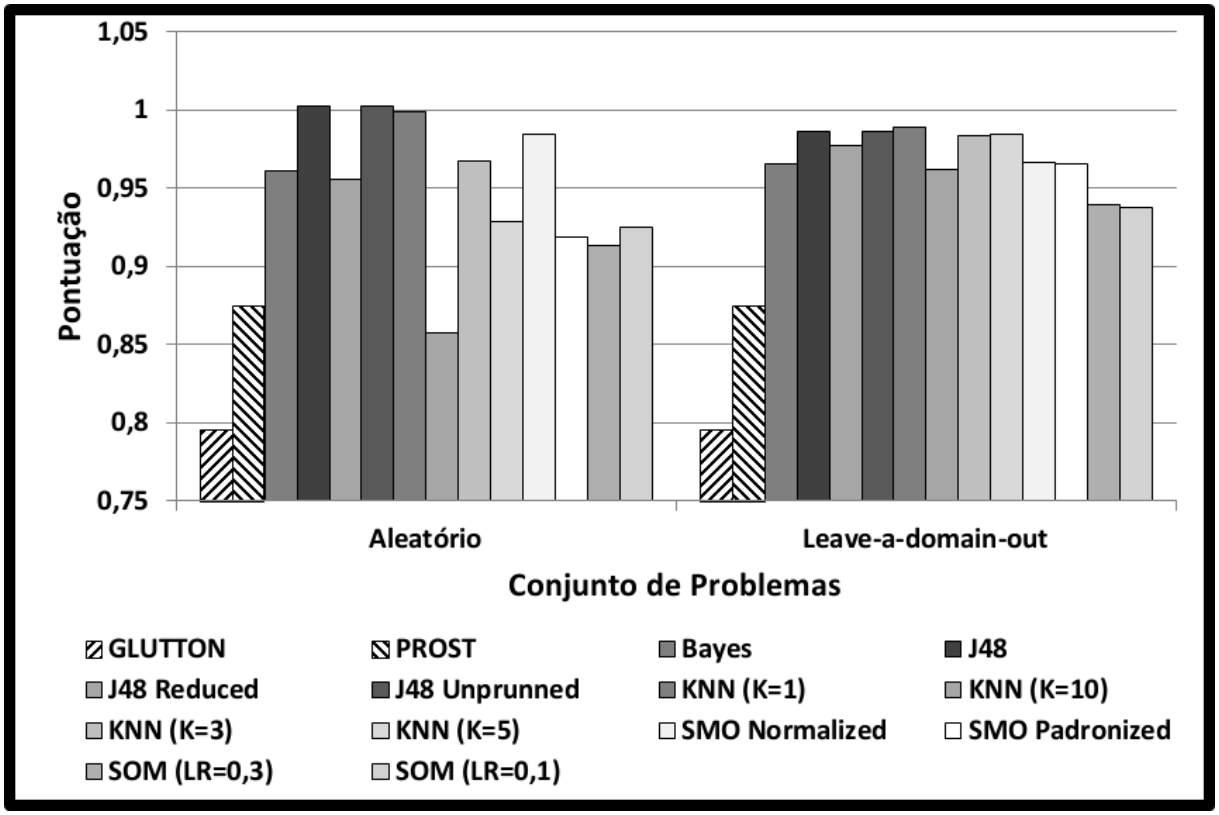

Figura 5.1: Resultados consolidados - classificadores, *PROST* e *Glutton*

Como é possível analisar na Figura 5.1, com exceção do planejador K-NN com $k = 10$

treinado no conjunto aleatório, todos os outros planejadores de predição superaram os resultados do *PROST* e do *Glutton*. O K-NN com $k = 10$ superou o resultado do planejador *Glutton*, mas perdeu para o planejador *PROST*. Predizer o melhor planejador também mostrou resultados efetivos no planejamento probabilístico, superando os planejadores probabilísticos com os melhores resultados na IPPC 2011.

Capítulo 6

Conclusão e Trabalhos Futuros

A dificuldade em propor novos algoritmos ou heurísticas motivou a busca por novas alternativas para resolver problemas de planejamento. Com os planejadores de portfólios se tornando eficientes, a composição dos portfólios com algoritmos de estratégias distintas se tornou o diferencial de sucesso. A proposta para se determinar os algoritmos do portfólio com melhor desempenho na solução de problemas e executá-los se mostrou eficiente nos testes apresentados, inclusive superando os planejadores de portfólio que conseguiram os melhores resultados nas últimas competições de planejamento, tanto em taxa de sucesso, como em tempo gasto para resolver o problema.

O presente trabalho também teve como objetivo exibir que a predição também se mostrou efetiva no planejamento probabilístico, superando os planejadores probabilísticos com os melhores resultados na IPPC 2011.

Como foi possível observar, a seleção das características corretas é um fator determinante na seleção e classificação, e que usar uma combinação grande de características não garante o sucesso da predição.

Os ambientes clássico e probabilístico possuem características e objetivos distintos. No ambiente clássico, é interessante minimizar o tempo para gerar um plano. Já no ambiente probabilístico, é necessário maximizar a pontuação encontrada para um determinado problema. Os classificadores se adequaram bem em ambas as propostas, tornando seu uso interessante para outras abordagens de planejamento. Como trabalhos futuros, são propostos:

- A utilização de técnicas de mineração de dados para determinação de novas características;
- Aplicação de classificação em planejamento clássico, objetivando minimizar o custo dos planos;
- Adição de avaliações multi-objetivas para minimizar tempo e custo em planejamento clássico;
- Aplicação das técnicas de predição nas outras linhas da competição de planejamento;

- Adição de outros algoritmos de classificação, problemas de planejamento e planejadores ao portfólio, e modificação dos parâmetros dos já pertencentes.

Referências Bibliográficas

[Aha e Kibler 1991] Aha, D. e Kibler, D. (1991). Instance-based learning algorithms. *Machine Learning*, 6:37–66.

[Alhossaini e Beck 2013] Alhossaini, M. A. e Beck, J. C. (2013). Instance-Specific Remodelling of Planning Domains by Adding Macros and Removing Operators. In Frisch, A. M. e Gregory, P. (editores), *SARA*. AAAI.

[Blum e Furst 1995] Blum, A. L. e Furst, M. L. (1995). Fast Planning Through Planning Graph Analysis. *ARTIFICIAL INTELLIGENCE*, 90(1):1636–1642.

[Bonet e Geffner 2001] Bonet, B. e Geffner, H. (2001). Planning as Heuristic Search. *Artificial Intelligence*, 129:5–33.

[Bonnet e Geffner 1998] Bonnet, B. e Geffner, H. (1998). HSP: Heuristic Search Planner. Entry at the AIPS-98 Planning Competition, Pittsburgh.

[Branquinho 2009] Branquinho, A. A. B. (2009). Planejamento de recursos para jogos de estratégia em tempo real. Master's thesis, Universidade Federal de Uberlândia.

[Burges 1998] Burges, C. J. C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Min. Knowl. Discov.*, 2(2):121–167.

[Cenamor et al. 2012] Cenamor, I., de la Rosa, T., e FernÃ¡ndez, F. (2012). Mining IPC-2011 Results. In *Proceedings of the Third Workshop on the International Planning Competition*, pp. 28–36.

[Coppin 2004] Coppin, B. (2004). *Artificial Intelligence Illuminated*. Jones and Bartlett Publishers, Inc., USA.

[de Souza Lima 2009] de Souza Lima, L. (2009). Class-Test: Classificação automática de testes para auxílio à criação de suítes de teste. Master's thesis, Universidade Federal de Pernambuco.

[Domshlak et al. 2010] Domshlak, C., Karpas, E., e Markovitch, S. (2010). To Max or not to Max: Online Learning for Speeding Up Optimal Planning.

[Edelkamp e Hoffmann 2003] Edelkamp, S. e Hoffmann, J. (2003). PDDL2.2: The Language for the Classical Part of the 4th International planning Competition. Technical report.

[Edelkamp e Schrödl 2012] Edelkamp, S. e Schrödl, S. (2012). *Heuristic Search - Theory and Applications*. Academic Press.

[Fikes e Nilsson 1971] Fikes, R. e Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artif. Intell.*, 2(3/4):189–208.

[Fink et al. 1998] Fink, E., Polya, G., e It, H. T. S. (1998). How to Solve It Automatically: Selection Among Problem-Solving Methods. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pp. 128–136. AAAI Press.

[Fox e Long 2003] Fox, M. e Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:2003.

[Furcy 2004] Furcy, D. A. (2004). *Speeding up the convergence of online heuristic search and scaling up offline heuristic search.* PhD thesis.

[Gerevini e Long 2005] Gerevini, A. e Long, D. (2005). Plan constraints and preferences in PDDL3 - the language of the fifth international planning competition. Technical report.

[Gerevini et al. 2009] Gerevini, A., Saetti, A., e Vallati, M. (2009). An Automatically Configurable Portfolio-based Planner with Macro-actions: PbP. In *ICAPS*.

[Gerevini et al. 2011] Gerevini, A., Saetti, A., e Vallati, M. (2011). PbP2: Automatic Configuration of a Portfolio-based Multi-Planner.

[Goldberg 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

[Hart et al. 1968] Hart, P., Nilsson, N., e Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107.

[Haykin 2001] Haykin, S. (2001). *Redes Neurais: princípios e prática.* Porto Alegre: Bookman, 2 edition.

[Helmert 2006] Helmert, M. (2006). The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26:191–246.

[Helmert e Gabriele 2011] Helmert, M. e Gabriele, R. (2011). Fast Downward Stone Soup : A Baseline for Building Planner Portfolios. In *ICAPS*.

[Hoey et al. 1999] Hoey, J., St-Aubin, R., Hu, A. J., e Boutilier, C. (1999). SPUDD: Stochastic Planning Using Decision Diagrams. In *Proceedings of Uncertainty in Artificial Intelligence*, Stockholm, Sweden.

[Hoffmann e Nebel 2001] Hoffmann, J. e Nebel, B. (2001). The FF planning system: fast plan generation through heuristic search. *J. Artif. Int. Res.*, 14(1):253–302.

[Hoos e Sttzle 2004] Hoos, H. e Sttzle, T. (2004). *Stochastic Local Search: Foundations & Applications.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Howe et al. 1999] Howe, A. E., Dahlman, E., Hansen, C., Scheetz, M., e Mayrhauser, A. V. (1999). Exploiting Competitive Planner Performance. In *In Proceedings of the Fifth European Conference on Planning*, pp. 62–72. Springer.

[Hunt et al. 1966] Hunt, E. B., Marin, J., e Stone, P. J. (1966). *Experiments in induction*. Academic Press.

[Keller e Eyerich 2012] Keller, T. e Eyerich, P. (2012). PROST: Probabilistic Planning Based on UCT. In *International Conference on Automated Planning and Scheduling*.

[Kohonen et al. 2000] Kohonen, T., Kaski, S., Lagus, K., Salojarvi, J., Paatero, V., e Saarela, A. (2000). Self Organization of a Massive Document Collection. *IEEE Transactions on Neural Networks*, 11:574–585.

[Kolobov et al. 2012] Kolobov, A., Dai, P., Daniel, M., e Weld, S. (2012). Reverse iterative deepening for finite-horizon MDPs with large branching factors. In *In International Conference on Automated Planning and Scheduling*.

[Korf 1990] Korf, R. E. (1990). Real-time heuristic search. *Artif. Intell.*, 42(2-3):189–211.

[Kotthoff 2012] Kotthoff, L. (2012). Algorithm Selection for Combinatorial Search Problems: A Survey. *CoRR*, abs/1210.7959.

[LaValle 2006] LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press, Cambridge, U.K. Available at http://planning.cs.uiuc.edu/.

[Malik et al. 2004] Malik, G., Nau, D., e Traverso, P. (2004). *Automated planning: theory and practice*. The Morgan Kaufmann Series in Artificial Intelligence Series. Elsevier/Morgan Kaufmann Publishers.

[Mcdermott et al. 1998] Mcdermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., e Wilkins, D. (1998). PDDL - The Planning Domain Definition Language. Technical Report TR-98-003, Yale Center for Computational Vision and Control,.

[Mitchell 1998] Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA.

[Neto et al. 2014] Neto, H. C., Julia, R. M. S., Caexeta, G. S., e Barcelos, A. R. A. (2014). LS-VisionDraughts: improving the performance of an agent for checkers by integrating computational intelligence, reinforcement learning and a powerful search method. *Applied Intelligence*, pp. 1–26.

[Pearl 1984] Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. The Addison-Wesley Series in Artificial Intelligence. Addison-Wesley.

[Pednault 1989] Pednault, E. P. D. (1989). ADL: exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, pp. 324–332, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Quinlan 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Raghavan et al. 2011] Raghavan, A. N., Joshi, S., Fern, A., e Tadepalli, P. (2011). Bidirectional Online Probabilistic Planning. `http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/Desc_Beaver.pdf`.

[Rice 1976] Rice, J. R. (1976). The Algorithm Selection Problem. *Advances in Computers*, 15:65–118.

[Richter e Westphal 2010] Richter, S. e Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks.

[Rintanen 2006] Rintanen, J. (2006). Introduction to Automated Planning. `http://users.ics.aalto.fi/rintanen/jussi/FreiburgAIPcourse/script.pdf`.

[Rish 2001] Rish, I. (2001). An empirical study of the naive Bayes classifier. In *International Joint Conference on Artificial Intelligence*, pp. 41–46. American Association for Artificial Intelligence.

[Roberts e Howe 2007] Roberts, M. e Howe, A. (2007). Learned models of performance for many planners. In *In ICAPS 2007, Workshop AI Planning and Learning.*

[Russell e Norvig 2009] Russell, S. J. e Norvig, P. (2009). *Artificial Intelligence: A Modern Approach.* Prentice Hall, 3rd edition.

[Sanner 2010] Sanner, S. (2010). Relational Dynamic Influence Diagram Language (RDDL): Language Description. `http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf`.

[Seipp et al. 2012] Seipp, J., Braun, M., Garimort, J., e Helmert, M. (2012). Learning Portfolios of Automatically Tuned Planners. In McCluskey, L., Williams, B., Silva, J. R., e Bonet, B. (editores), *ICAPS*. AAAI.

[Tsoumakas et al. 2004] Tsoumakas, G., Vrakas, D., Bassiliades, N., e Vlahavas, I. (2004). Using the k nearest problems for adaptive multicriteria planning. In *in Proceedings of the 3rd Hellenic Conference on Artificial Intelligence, SETN04*, pp. 132–141. Springer.

[Ure et al. 2012] Ure, K., Geramifard, A., Chowdhary, G., e How, J. P. (2012). Adaptive Planning for Markov Decision Processes with Uncertain Transition Models via Incremental Feature Dependency Discovery. In *European Conference on Machine Learning (ECML).*

[Valenzano et al. 2012] Valenzano, R., Nakhost, H., Müller, M., Schaeffer, J., e Sturtevant, N. (2012). ArvandHerd: Parallel Planning with a Portfolio. *European Conference on Artificial Intelligence (ECAI 2012).*

[Vallati e Kitchin 2012] Vallati, M. e Kitchin, D. E. (2012). Challenges of Portfolio-based Planning. In *30th Workshop of the UK Planning and Scheduling Special Interest Group*, pp. 1–4.

[Vrakas et al. 2003] Vrakas, D., Tsoumakas, G., Bassiliades, N., e Vlahavas, I. (2003). Learning Rules for Adaptive Planning. In *In Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS03)*, pp. 82–91.

[Weld 1999] Weld, D. S. (1999). Recent Advances in AI Planning. *AI Magazine*, 20(2):93–123.

[Witten e Frank 2005] Witten, I. H. e Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Wolpert e Macready 1997] Wolpert, D. H. e Macready, W. G. (1997). No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82.

[Xie et al. 2012] Xie, F., Nakhost, H., e Müller, M. (2012). Planning Via Random Walk-Driven Local Search. In *ICAPS*.