Descoberta de regras de conhecimento
utilizando computação evolutiva multiobjetivo

Rafael Giusti

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Descoberta de regras de conhecimento utilizando computação evolutiva multiobjetivo

Rafael Giusti

Orientador: *Prof. Dr. Gustavo Enrique Almeida Prado Alves Batista*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional.

USP – São Carlos – Maio/2010

Trabalho desenvolvido com financiamento FAPESP, processo Nº 2007/06305-7.

Resumo

Na área de inteligência artificial existem algoritmos de aprendizado, notavelmente aqueles pertencentes à área de aprendizado de máquina – AM –, capazes de automatizar a extração do conhecimento implícito de um conjunto de dados. Dentre estes, os algoritmos de AM simbólico são aqueles que extraem um modelo de conhecimento inteligível, isto é, que pode ser facilmente interpretado pelo usuário. A utilização de AM simbólico é comum no contexto de classificação, no qual o modelo de conhecimento extraído é tal que descreve uma correlação entre um conjunto de atributos denominados premissas e um atributo particular denominado classe.

Uma característica dos algoritmos de classificação é que, em geral, estes são utilizados visando principalmente a maximização das medidas de cobertura e precisão, focando a construção de um classificador genérico e preciso. Embora essa seja uma boa abordagem para automatizar processos de tomada de decisão, pode deixar a desejar quando o usuário tem o desejo de extrair um modelo de conhecimento que possa ser estudado e que possa ser útil para uma melhor compreensão do domínio.

Tendo-se em vista esse cenário, o principal objetivo deste trabalho é pesquisar métodos de computação evolutiva multiobjetivo para a construção de regras de conhecimento individuais com base em critérios definidos pelo usuário. Para isso utiliza-se a biblioteca de classes e ambiente de construção de regras de conhecimento ECLE, cujo desenvolvimento remete a projetos anteriores. Outro objetivo deste trabalho consiste comparar os métodos de computação evolutiva pesquisados com métodos baseado em composição de *rankings* previamente existentes na ECLE.

É mostrado que os métodos de computação evolutiva multiobjetivo apresentam melhores resultados que os métodos baseados em composição de *rankings*, tanto em termos de dominância e proximidade das soluções construídas com aquelas da fronteira Pareto-ótima quanto em termos de diversidade na fronteira de Pareto. Em otimização multiobjetivo, ambos os critérios são importantes, uma vez que o propósito da otimização multiobjetivo é fornecer não apenas uma, mas uma gama de soluções eficientes para o problema, das quais o usuário pode escolher uma ou mais soluções que apresentem os melhores compromissos entre os objetivos.

Abstract

Machine Learning algorithms are notable examples of Artificial Intelligence algorithms capable of automating the extraction of implicit knowledge from datasets. In particular, Symbolic Learning algorithms are those which yield an intelligible knowledge model, *i.e.*, one which a user may easily read. The usage of Symbolic Learning is particularly common within the context of classification, which involves the extraction of knowledge such that the associated model describes correlation among a set of attributes named the premises and one specific attribute named the class.

Classification algorithms usually target into creating knowledge models which maximize the measures of coverage and precision, leading to classifiers that tend to be generic and precise. Although this constitutes a good approach to creating models that automate the decision making process, it may not yield equally good results when the user wishes to extract a knowledge model which could assist them into getting a better understanding of the domain.

Having that in mind, it has been established as the main goal of this Master's thesis the research of multi-objective evolutionary computing methods to create individual knowledge rules maximizing sets of arbitrary user-defined criteria. This is achieved by employing the class library and knowledge rule construction environment ECLE, which had been developed during previous research work. A second goal of this Master's thesis is the comparison of the researched evolutionary computing methods against previously existing ranking composition methods in ECLE.

It is shown in this Master's thesis that the employment of multi-objective evolutionary computing methods produces better results than those produced by the employment of ranking composition-based methods. This improvement is verified both in terms of solution dominance and proximity of the solution set to the Pareto-optimal front and in terms of Pareto-front diversity. Both criteria are important for evaluating the efficiency of multi-objective optimization algorithms, for the goal of multi-objective optimization is to provide a broad range of efficient solutions, so the user may pick one or more solutions which present the best trade-off among all objectives.

Agradecimentos

Agradeço, antes de considerar qualquer nome, àqueles que sempre estiveram ao meu lado: meus pais, Wilson e Regina, que têm observado diariamente a “briga” que venho travando nos últimos meses para chegar ao ponto de realizar este agradecimento. Agradeço também ao meu irmão Daniel, de minha avó Rosa e minha tia Simone, que me influenciaram tanto em tudo. Foi com esse pequeno grupo que eu aprendi a enxergar a mim e a todas as outras pessoas com quem convivo.

Sou muito grato ao meu orientador, Professor Gustavo Batista, que tem acreditado em mim, me orientado e me incentivado há mais de três anos, desde meu projeto de conclusão de curso. Aprendi muito com ele, não apenas academicamente, mas também pessoal e profissionalmente. E finalizo este trabalho de Mestrado com a confiança de que estive acompanhado não apenas de um professor, mas também de um amigo.

Agradeço à Professora Carolina Monard e ao Adriano Pila. A Professora Carolina foi minha primeira orientadora e o Adriano foi meu primeiro colega de trabalho e um “co-orientador extraoficial”. Além da oportunidade de ingressar no mundo acadêmico e de ter podido “herdar” a ECLE, ganhei muita experiência acadêmica e profissional durante aquele projeto de IC.

Agradeço ao Bruno Nogueira, à Cláudia Milaré, ao Fabiano Fernandes, ao Ígor Braga, ao Rafael Rossi, ao Ricardo Marcacini ao Professor Ronaldo Prati, à Rosa del Gaudio e à Professora Solange Rezende por terem me auxiliado diretamente na execução deste trabalho, seja com a revisão de algum texto ou relatório, com a leitura de algum capítulo da Dissertação, auxiliando nas etapas de implementação e avaliação ou me ajudando a colocar a cabeça no lugar nos momentos em que tudo parecia muito difícil. Agradeço também a todos os meus colegas e amigos do Laboratório de Inteligência Computacional e de outros laboratórios, André, Andrés, Chino, “Dieguito”, Everton, Jean, Matheus “Caneca”, Merley, Nathalie, Nils, Tatiane, Thyago e Victor, com quem convivi e compartilhei tantas boas experiências.

Sumário

Resumo	7
Abstract	9
Agradecimentos	11
Sumário	13
Lista de Figuras	17
Lista de Tabelas	19
1 Introdução	21
1.1 Contextualização e Motivação	21
1.2 Objetivos e Hipótese	23
1.3 Organização do Texto	24
2 Aprendizado de Máquina	25
2.1 Considerações Iniciais	25
2.2 Inteligência Artificial: Visão Geral	26
2.3 Modalidades de AM	28
2.3.1 Aprendizado Supervisionado e Não-Supervisionado	28
2.3.2 Aprendizado Simbólico e Não-Simbólico	30
2.4 Terminologia e Linguagem de Descrição	31
2.4.1 Tabela Atributo-Valor	31
2.4.2 Regras de Conhecimento	33
2.4.3 Matriz de Contingência	35
2.5 Qualidade de Regras	37

2.5.1	Medidas de Qualidade de Regras	38
2.5.2	Medidas de Interesse	42
2.6	Considerações Finais	47
3	Otimização	49
3.1	Considerações Iniciais	49
3.2	Otimização Simples-Objetivo	49
3.3	Otimização Multiobjetivo	53
3.4	<i>Rankings</i> para Otimização Multiobjetivo	55
3.5	Composição de <i>Rankings</i>	57
3.5.1	Médias aritmética e harmônica ponderadas	59
3.5.2	Mediana	60
3.5.3	O <i>Ranking</i> de Condorcet	60
3.5.4	O <i>Ranking</i> Recíproco	62
3.6	Métodos de Resolução	62
3.7	Considerações Finais	67
4	Computação Evolutiva e Algoritmos Evolutivos	69
4.1	Considerações Iniciais	69
4.2	Computação Evolutiva	70
4.3	Algoritmos Evolutivos	71
4.4	Indivíduos	72
4.5	Função de Avaliação	75
4.6	Operador de Seleção	77
4.6.1	A Roda da Roleta	78
4.6.2	Seleção por <i>ranking</i>	80
4.6.3	O Torneio	81
4.7	Operador de <i>Crossover</i>	83
4.8	Operador de Mutação	85
4.9	Outras Considerações	86
4.10	Considerações Finais	88
5	Ambiente Computacional	89
5.1	Considerações Iniciais	89
5.2	A Biblioteca de Classes DOL	90
5.3	A Biblioteca de Classes ECLE	91

5.3.1	Indivíduos	93
5.3.2	Função de Avaliação e Operador de Seleção	97
5.3.3	Operadores de <i>Crossover</i>	98
5.3.4	Operadores de Mutação	102
5.4	Algoritmos Evolutivos Implementados na ECLE	106
5.4.1	Algoritmo Evolutivo Simples-Objetivo Baseado em <i>Rankings</i>	108
5.4.2	Vector Evaluated Genetic Algorithm	109
5.4.3	Elitist Non-Dominated Sorting Genetic Algorithm	110
5.5	Considerações Finais	112
6	Avaliação Experimental	115
6.1	Considerações Iniciais	115
6.2	Crerios de Comparação	116
6.2.1	Medidas de Dominância	118
6.2.2	Medidas de Diversidade	123
6.3	Configuração dos Experimentos	126
6.4	Resultados	128
6.5	Considerações Finais	131
7	Conclusão	133
7.1	Resumo do Projeto e Revisão do Trabalho	133
7.2	Principais Contribuições	134
7.3	Limitações e Trabalhos Futuros	135
	Referências	137

Lista de Figuras

2.1 Exemplo de regra de conhecimento relacionada à base de dados <i>tênis</i> (apresentada na Tabela 2.2)	34
2.2 Exemplo de regra de conhecimento com conjunção.	34
2.3 Exemplo de regras pouco úteis relacionadas à base de dados <i>tênis</i> . . .	37
3.1 Exemplo de espaço de decisão com solução ótima indicada.	50
3.2 Exemplo de espaço de decisão multiobjetivo	53
3.3 Duas fronteiras de Pareto para um problema de otimização (max,max) .	55
3.4 Exemplo de busca no espaço de decisão.	64
3.5 Exemplo de transição do <i>hill climbing</i>	64
3.6 Exemplo de transição de um algoritmo de otimização multiobjetivo. . . .	65
3.7 Exemplo de uma função objetivo com máximos e mínimos locais.	66
4.1 Exemplo de sobreposição de regras	74
4.2 Método de seleção roda da roleta	79
4.3 Ilustração do <i>crossover</i> de um ponto.	84
4.4 Ilustração de um operador de mutação.	85
5.1 Arquitetura de pacotes da ECLE.	92
5.2 Exemplo de regra de conhecimento no formato <i>PBM</i>	94
5.3 Conceituação geral de uso típico da ECLE	95
5.4 Codificação da regra de conhecimento	95
5.5 Exemplo de regra <i>PBM</i> de um domínio de concessão de crédito.	96
5.6 Exemplo de codificação da regra de conhecimento	97
5.7 Exemplo de aplicação do <i>crossover</i> estrutural.	99
5.8 Exemplo de aplicação do <i>crossover</i> de atributo.	100
5.9 Exemplo de aplicação do <i>crossover</i> local.	101

5.10 Exemplo de mutação estrutural.	103
5.11 Exemplo de mutação estrutural com inserção de condição	104
5.12 Exemplo de aplicação do operador de mutação local.	105
6.1 Dois conjuntos de soluções pertencentes ao mesmo espaço de objetivos, com suas respectivas fronteiras de Pareto destacadas.	117
6.2 Regras representadas no espaço de objetivos.	120
6.3 Soluções únicas não-dominadas.	122
6.4 Regras referentes à base breast.	125

Lista de Tabelas

2.1	Formato geral da tabela atributo-valor	31
2.2	Conjunto de dados <i>tênis</i> representado como tabela -atributo-valor . . .	33
2.3	Exemplo de matriz de contingência.	37
2.4	Exemplo de matriz de contingência.	37
2.5	Sumário das medidas de interesse.	48
3.1	Exemplo de itens relevantes para o problema da mochila.	52
3.2	<i>Ranking</i> envolvendo 10 indivíduos.	56
3.3	<i>Ranking</i> posicional.	56
3.4	<i>Ranking</i> normalizado.	57
3.5	Exemplo de composição de <i>rankings</i> pelo método da média aritmética. .	58
3.6	<i>Ranks</i> atribuídos a uma solução I_i qualquer.	60
3.7	Soluções comparadas pelo <i>ranking</i> de Condorcet.	61
3.8	Vitórias e empates contabilizados para as soluções da Tabela 3.7	61
3.9	Composição final do <i>ranking</i> de Condorcet.	62
3.10	Exemplo completo do processo de composição por <i>ranking</i> recíproco . .	62
4.1	Três operadores de seleção aplicados a uma população, exemplificando três diferentes cenários de pressão de seleção.	78
5.1	Reprodução da Tabela 2.2, apresentando o conjunto de dados <i>tênis</i> . .	94
6.1	Medidas para os conjuntos de soluções apresentados na Figura 6.2. . .	120
6.2	Medidas para os conjuntos de soluções apresentados na Figura 6.3. . .	122
6.3	Medidas de diversidade para os conjuntos apresentados na 6.4.	125
6.4	Informações sobre os conjuntos de dados usados. A última coluna indica o número de classes de cada conjunto de dados e o número de exemplos em cada classe – os nomes das classes são omitidos.	126

6.5	Medidas utilizadas nos diferentes cenários.	127
6.6	Médias de medidas de dominância, diversidade e tamanho de população para todos os conjuntos e cenários de experimentos.	128
6.7	Médias de todas as execuções experimentais agrupadas por métodos. . .	129
6.8	Resultados do teste de hipótese para dominância.	129
6.9	Resultados do teste de hipótese para diversidade.	130

Introdução

1.1 Contextualização e Motivação

Há anos, acumular dados tem sido parte das decisões estratégicas de empresas e organizações ao redor de todo o mundo. Bases de dados com centenas de *gigabytes* ou mesmo *terabytes* de tamanho têm se tornado cada vez mais comuns, pois os dados acumulados durante anos de trabalho guardam implicitamente um conhecimento que pode ser valioso para a compreensão de algum processo ou para a obtenção de vantagem competitiva. Por exemplo, se um analista financeiro trabalhou durante anos na concessão de empréstimos, então os dados por ele registrados devem conter implicitamente as direções que o levaram à tomada de decisão (concessão ou não do empréstimo). Dessa forma, uma transformação desses dados em conhecimento pode proporcionar auxílio inteligente à tomada de decisões.

Na área de inteligência artificial existem algoritmos de aprendizado, notavelmente aqueles pertencentes à área de aprendizado de máquina – AM –, capazes de automatizar a extração do conhecimento implícito de um conjunto de dados ([Quinlan, 1988](#); [Clark e Boswell, 1991](#); [Monard e Baranauskas, 2003a,b](#)). Em AM, os dados são tipicamente estruturados na forma de exemplos, sendo cada exemplo um conjunto de valores referentes a um conjunto de atributos. Possivelmente, cada exemplo representa uma situação ou conceito real, como uma lista de itens adquiridos por um cliente de um mercado, os sintomas apresentados por um paciente diagnosticado etc. O conhecimento extraído por um algoritmo de AM é um modelo que representa, de forma aproximada, algum conceito que pode ser generalizado dos dados.

Existem diversas modalidades em AM e diversas formas de se classificar algoritmos de AM. Uma dessas formas diz respeito à inteligibilidade do modelo extraído e outra diz respeito à natureza dos exemplos.

No que diz respeito à inteligibilidade, o algoritmo pode ser considerado simbólico ou não-simbólico. Algoritmos de AM simbólico são aqueles que representam o conheci-

mento extraído de forma inteligível para o usuário — *e.g.* regra de conhecimento (Battista et al., 2006). A aplicação de AM simbólico é particularmente interessante quando o usuário deseja automatizar um processo de tomada de decisão e, ao mesmo tempo, compreender os conceitos envolvidos nesse processo ou ainda quando o usuário tem por objetivo a extração de um modelo que possa ser estudado para potencialmente inspirar descobertas.

No que diz respeito à natureza dos exemplos, o aprendizado pode ser supervisionado, semissupervisionado ou não-supervisionado. Quando o aprendizado é supervisionado, um ou mais atributos são tais que, em cada exemplo, os valores desses atributos assumem uma relação de causalidade com os atributos restantes. No caso mais específico de classificação existe um único atributo, chamado classe ou rótulo, tal que seu valor pode ser previsto com base nos valores dos outros atributos. O modelo extraído pelo algoritmo de AM é chamado classificador e pode ser simbólico ou não.

Uma característica dos algoritmos de classificação é que, em geral, estes são desenvolvidos com o objetivo principal de extrair classificadores que sejam genéricos e precisos. Isto é, o algoritmo procura construir classificadores que possam ser aplicados ao maior número de exemplos possível e que sejam capazes de predizer as classes corretamente. Essa abordagem é bastante eficiente quando o interesse reside na automatização do processo de tomada de decisão, mas pode ser insuficiente quando o usuário deseja adquirir para si o conhecimento extraído dos dados, pois desconsidera outras propriedades interessantes, além da cobertura e da precisão, como a dependência entre o corpo e a cabeça das regras, a descoberta de exceções e a relação entre a precisão das regras e a distribuição dos valores de classes nos exemplos.

Tendo-se em vista esse cenário, torna-se interessante potencializar a descoberta de conhecimento por outros critérios, os quais possivelmente permitam que o usuário obtenha modelos de conhecimento que sejam mais úteis ou interessantes para si. Em sua tese de Doutorado, Pila (2007) propõe e, em conjunto com Giusti et al. (2006), desenvolve¹ um método para construir regras de conhecimento individuais com base em diferentes critérios simultâneos definidos pelo usuário. A esses critérios o proponente deu o nome *propriedades específicas*, as quais são aferidas por medidas de qualidade de regra como cobertura, precisão, novidade, *lift* e outras. A construção de regras com propriedades específicas foi abordada com a utilização de um método de computação evolutiva baseado em composição de *rankings* que, como será explicado na Seção 3.4, constitui uma forma de se utilizar otimização simples-objetivo para a resolução de um problema de otimização multiobjetivo. Durante a execução do projeto os autores também desenvolveram um ambiente e uma biblioteca de classes para descoberta de conhecimento com propriedades específicas, a ECLE – *Evolutionary Computing Learning Environment*.

¹A ECLE, bem como a abordagem de construção de regras de conhecimento com “propriedades específicas”, foi proposta por (Pila, 2007) em seu trabalho de Doutorado. O autor desta Dissertação participou em projetos de iniciação científica que tinha interseção com esse trabalho de Doutorado, sendo co-autor da ECLE, mas não de sua proposta.

Desde as primeiras aplicações a ECLE tem se mostrado bastante eficiente na construção de regras com base em critérios especificados pelo usuário (Pila et al., 2006c; Batista et al., 2007; Giusti et al., 2008) e também como um *framework* para construção de regras utilizando computação evolutiva. Apesar disso, a ausência de uma abordagem intrinsecamente multiobjetiva levanta dúvidas se tal abordagem poderia prover melhores resultados. Por esse motivo, os autores estabeleceram como trabalho futuro a incorporação de métodos de otimização multiobjetivo à ECLE.

É relevante ressaltar aqui que, embora a proposta de utilização de composição de *rankings* para a construção de regras individuais com base em propriedades específicas tenha partido do então doutorando Adriano D. Pila, o autor desta Dissertação participou ativamente no desenvolvimento da ECLE em projetos de iniciação científica² e hoje é seu mantenedor.

1.2 Objetivos e Hipótese

Considerando-se o cenário apresentado na seção anterior, os seguintes objetivos foram definidos para este trabalho de mestrado:

- A investigação de técnicas de otimização multiobjetivo, baseadas no conceito de eficiência de Pareto, para a construção de regras de conhecimento com critérios diversos especificados pelo usuário;
- A modificação da ECLE para a utilização de métodos de composição de *rankings*, previamente existentes, em um contexto de otimização multiobjetivo;
- A comparação da eficácia de métodos de composição de *rankings* contra a eficácia de métodos de computação evolutiva multiobjetivo baseados nos conceitos de eficiência de Pareto.

Tais objetivos podem ser justificados pela definição da seguinte hipótese:

“A utilização de métodos específicos de otimização multiobjetivo possibilita a construção de regras de conhecimento que atendam critérios múltiplos de qualidade definidos pelo usuário.”

Considerando-se que Pila (2007) demonstrou que a abordagem baseada em composição de *rankings* é capaz de construir regras de conhecimento que atendam às propriedades específicas, a demonstração de que os métodos baseados em otimização multiobjetivo superam os métodos baseados em composição de *rankings* comprova

²O autor participou de dois projetos de iniciação científica entre os anos de 2005 e 2007 envolvendo construção de regras de conhecimento utilizando computação evolutiva. Em ambos os projetos o autor utilizou e modificou a ECLE na condução da pesquisa. Ambos os processos foram financiados pela FAPESP – processos 2005/01178-1 e 2007/01259-7.

a hipótese. A hipótese foi assim verificada como válida e evidências empíricas que apoiam essa conclusão são apresentadas no Capítulo 6 desta Dissertação.

Considera-se que os objetivos assinalados nesta seção constituem as principais contribuições desta Dissertação. No Capítulo 7 essas contribuições são discutidas em maiores detalhes.

1.3 Organização do Texto

O restante desta Dissertação está organizado como descrito a seguir.

No Capítulo 2 é feita uma discussão sobre inteligência artificial e aprendizado de máquina. Conceitos fundamentais para a compreensão do trabalho realizado são explicados. Alguns critérios que podem ser definidos, como medidas de qualidade e de interesse de regras de conhecimento, são apresentados neste capítulo.

No Capítulo 3 são apresentados fundamentos de otimização e uma discussão sobre otimização simples-objetivo e otimização multiobjetivo é realizada, tendo como principal objetivo introduzir os conceitos de eficiência de Pareto e fronteira de Pareto. Os conceitos introduzidos neste capítulo são especialmente importantes para a compreensão do Capítulo 4 e também para a compreensão da distinção entre métodos especificamente voltados para otimização multiobjetivo e métodos baseados na composição de *rankings*.

No Capítulo 4 é apresentada a área de computação evolutiva, com especial atenção aos algoritmos genéticos. A proposta deste trabalho envolve a utilização de um algoritmo genético multiobjetivo, portanto os conceitos apresentados neste capítulo e nos capítulos anteriores são fundamentais para a compreensão do trabalho.

No Capítulo 5 é descrita a ECLE – *Evolutionary Computing Learning Environment*. A ECLE é uma biblioteca de classes e um ambiente de aprendizado de máquina baseado em computação evolutiva multiobjetivo. Todas as propostas deste trabalho foram implementadas como funcionalidades da ECLE. A ECLE também foi utilizada para realizar a avaliação do método proposto.

No Capítulo 6 é descrito o plano de avaliação experimental e os resultados experimentais obtidos. Uma discussão sobre os resultados obtidos e sobre resultados esperados pode ser encontrada neste capítulo.

Finalmente, no Capítulo 7 é feita a conclusão deste texto. As principais contribuições, na visão do autor, são destacadas e uma autocrítica com relação aos limites deste projeto de pesquisa também é realizada. Algumas metas para trabalho futuros são traçadas e justificadas.

Aprendizado de Máquina

2.1 Considerações Iniciais

Neste capítulo é realizada uma breve introdução ao aprendizado de máquina. Esta introdução não possui o intuito de ser completa, ou seja, de descrever todos os principais paradigmas de aprendizado de máquina, mas definir conceitos e notações que serão utilizados durante o restante da Dissertação e realizar uma breve introdução à área. O foco principal deste capítulo reside no aprendizado simbólico, o qual induz hipóteses que podem ser facilmente lidas por seres humanos. Tais hipóteses são frequentemente descritas na forma de regras de decisão.

Um outro conceito importante que será descrito nesse capítulo é a avaliação de regras de decisão por meio do *framework* de Lavrač (Lavrač et al., 1999), o qual permite utilizar as matrizes de contingência das regras para avaliá-las segundo diferentes medidas. Essas medidas são importantes para este trabalho, pois permitem verificar se as regras atendem a propriedades que podem ser de interesse do usuário. Entretanto, tais medidas não representam a única forma de avaliação de regras que pode ser associada a este trabalho.

Este capítulo está organizado da seguinte maneira. Na Seção 2.2 é apresentada uma visão geral sobre a inteligência artificial e aprendizado de máquina. Na Seção 2.3 são introduzidos os conceitos de aprendizado supervisionado e não-supervisionado, bem como aprendizado simbólico e não-simbólico. Na Seção 2.4 é descrita a notação, bem como a terminologia adotada neste trabalho para descrever regras de decisão. Finalmente, na Seção 2.5 é descrito o *framework* de Lavrač e algumas medidas de qualidade que podem ser descritas a partir desse *framework*.

2.2 Inteligência Artificial: Visão Geral

A capacidade de aprender não é uma habilidade exclusiva ao ser humano, mas é certamente um dos atributos com os quais nos identificamos mais facilmente, ao lado de alguns outros como a criatividade e a habilidade de prever resultados de ações e eventos que ainda estão em andamento. De acordo com o Moderno Dicionário da Língua Portuguesa Michaelis, aprendizado é o processo por meio do qual uma entidade incorre mudanças permanentes de comportamento como resultado de treino ou experiência anterior. Intuitivamente, todos temos familiaridade com essa definição. Sempre que estudamos ou buscamos mudanças de atitude ou de objetivo pela análise de experiências, estamos aprendendo.

O estudo científico da natureza do aprendizado faz parte das atribuições da neuropsicologia. Bloom et al. (1956) define uma taxonomia que atualmente é conhecida como *taxonomia de Bloom*. Embora tenha passado por mudanças estruturais em virtude dos avanços da neuropsicologia, em sua essência a taxonomia de Bloom separa o aprendizado em três domínios, a saber, o domínio *cognitivo*, o domínio *afetivo* e o domínio *psicomotor*. Sucintamente, os três domínios estão relacionados principalmente à aquisição do conhecimento – o “saber como” e “saber o quê” –, ao desenvolvimento emocional relacionado à compreensão do sentimento individual e ao desenvolvimento da destreza e da habilidade em utilizar ferramentas.

O desejo de construir “máquinas inteligentes”, capazes de adquirir conhecimento por conta própria, vem acompanhando a humanidade há séculos. A mitologia está repleta de criaturas artificiais que apresentam traços de inteligência, como o golem – uma criatura de barro antropomórfica com traços de inteligência reduzidos. No século XX, com os adventos da computação e da robótica, a busca por “máquinas inteligentes” deixou de ser apenas ficção e mitologia e tornou-se ciência, especialmente com o advento da inteligência artificial.

O termo *inteligência artificial* – IA – refere-se a um amplo campo de pesquisa multidisciplinar que combina, entre outras áreas, computação, matemática e ciência cognitiva no desenvolvimento de sistemas que demonstrem “comportamento inteligente” ou que solucionem problemas cujas resoluções tipicamente envolvem “o uso da inteligência”. Os primeiros passos da IA foram dados entre as décadas de 1930 e 1950, fomentados inicialmente por avanços da neurociência que levaram à compreensão do neurônio como um mecanismo de transmissão de informação digital. Por volta da mesma época, os estudos de Alan Turing e a formalização do modelo matemático que ficaria depois conhecido como a *máquina de Turing* (Turing, 1937) mostraram que qualquer processo de cômputo pode ser descrito de forma digital. Em 1946, na *Dartmouth Summer Research Conference on Artificial Intelligence*, influenciado pelos avanços combinados da neurociência e da teoria da computação, John McCarthy propôs o termo “inteligência artificial” e sugeriu que:

“Todo aspecto do aprendizado ou qualquer outra característica da inteligência pode ser, em princípio, descrita de forma tão precisa que uma máquina pode ser construída para simular tais aspectos e características.”

Durante anos, o objetivo principal da IA foi a construção de “máquinas pensantes”, capazes de exercer aprendizado com um potencial humano ou quase-humano. O otimismo inicial era tão grande que muitos pesquisadores sustentavam a hipótese de que computadores logo seriam capazes de descobrir e desenvolver quaisquer teoremas e hipóteses automaticamente. Com o tempo, a dificuldade em alcançar tais resultados mudou a visão dos pesquisadores e linhas de pesquisa mais “realistas” emergiram ([Russell e Norvig, 2003](#)). Traçando um paralelo com a taxonomia de Bloom, pode-se dizer que a pesquisa em IA deixou de buscar um sistema capaz de aprender em todos os aspectos que um ser humano é capaz de aprender e se subdividiu em diversas linhas especializadas em apenas alguns aspectos dos três domínios de Bloom. Por exemplo, atualmente existem pesquisas que concentram-se exclusivamente na construção de robôs antropomórficos capazes de andar como humanos, ou na construção de dispositivos mecatrônicos antropomórficos que se comportem como partes do corpo humano, a exemplo de braços mecânicos para deficientes físicos – domínio psicomotor. Outras linhas de pesquisa buscam a construção de “robôs sociais” que expressem sentimentos humanos ou que se comuniquem como seres humanos – domínio afetivo. Outras linhas de pesquisa estudam apenas a aquisição de conhecimento e programas capazes de absorver informação nova de forma automática – domínio cognitivo. A linha de pesquisa de aprendizado de máquina se insere na última categoria.

Aprendizado de máquina – AM – é uma linha de pesquisa em IA que tem por objetivo o estudo de técnicas relacionadas ao aprendizado e o desenvolvimento de sistemas capazes de aprender automaticamente. A premissa básica do aprendizado de máquina é que os dados contêm implicitamente conhecimento relacionado a algum processo que explica esses dados. Por exemplo, o histórico de um paciente, anotado por médicos ao longo dos anos, pode conter conhecimento implícito que explique as condições médicas desse paciente. De forma semelhante, uma coleção de históricos médicos pode permitir um estudo aprofundado sobre certas condições médicas, permitindo a descoberta de conhecimento possivelmente inédito sobre essas condições. Ainda que não seja possível determinar com exatidão o processo relacionado aos dados, é possível encontrar uma aproximação desse processo. O papel de um sistema de aprendizado é construir um modelo capaz de explicar, com precisão limitada, o processo relacionado aos dados, utilizando para isso métodos estatísticos que se baseiam em inferências sobre um subconjunto – amostra – dos dados ([Alpaydin, 2004](#)).

A transformação de dados em conhecimento vem sendo realizada há séculos. No século XVI Johannes Kepler (1571—1630) estudou o movimento aparente do planeta Marte a partir de dados coletados por Tycho Brahe (1546—1601), do qual formulou três hipóteses sobre o movimento dos astros que hoje conhecemos como as três leis

de Kepler. As análises de Kepler, entretanto, podem ser consideradas “artesaniais” por não terem sido realizadas segundo um método sistemático. Progressos posteriores, como o teorema de Bayes (séc. XVIII) e o surgimento da análise de regressão (séc. XIX) estabeleceram técnicas sistemáticas de transformação de dados em conhecimento, os quais certamente facilitaram a transformação manual. A partir do século XX, entretanto, o grande volume de dados e a urgência crescente de se produzir conhecimento tornou o processo manual inviável (Garcia et al., 2003). Atualmente, o processo de *extração de conhecimento de bases de dados* – KDD, do inglês *Knowledge Discovery in Databases* – utiliza diversas técnicas, entre elas o AM, para construir automaticamente modelos de conhecimento a partir dos dados.

O processo de KDD pode ser realizado com diferentes finalidades, resultando na construção de modelos que podem ser preditivos, descritivos ou ambos. Modelos *preditivos* são construídos e utilizados com o objetivo de aplicar o conhecimento extraído na previsão de casos esperados ou na detecção de irregularidades. Por exemplo, um banco pode utilizar redes neurais para construir um modelo capaz de reconhecer transações de crédito incomuns e potencialmente fraudulentas, permitindo que milhões de transações sejam avaliadas automaticamente todos os dias – o que seria impossível se dependesse de análise manual. Modelos *descritivos*, por sua vez, são interessantes quando o processo de KDD é utilizado para auxiliar a compreensão dos dados. Por exemplo, o uso de KDD em uma coleção de laudos médicos pode auxiliar o pesquisador a descobrir correlações até então desconhecidas entre sintomas e doenças, o que pode impulsionar a pesquisa médica e incentivar os pesquisadores a realizar novas descobertas.

2.3 Modalidades de AM

Existem várias técnicas ou modalidades de AM, as quais podem ser classificadas segundo diferentes critérios. Com relação à natureza dos dados utilizados pelos métodos de AM, por exemplo, o aprendizado pode ser supervisionado, não-supervisionado ou semissupervisionado. Com relação à natureza do modelo construído, o aprendizado pode ser simbólico ou não-simbólico. Tais critérios são descritos nas subseções seguintes.

2.3.1 Aprendizado Supervisionado e Não-Supervisionado

No processo de KDD, as técnicas de AM podem ser classificadas entre aprendizado supervisionado, não-supervisionado e semissupervisionado, dependendo da natureza dos exemplos utilizados. Em todos os casos, o conjunto de dados de treinamento é um conjunto de exemplos $E = \{E_1, E_2, \dots, E_N\}$ e cada exemplo $E_i = (\mathbf{x}_i, y_i)$ é composto por um vetor de valores $\mathbf{x}_i$ e um valor de classe y_i . O valor de classe y_i pode ser

definido para todos os exemplos, apenas para um subconjunto dos exemplos ou pode não ser definido para nenhum dos exemplos.

Quando o valor de classe é definido para todos os exemplos, a modalidade de AM utilizada é chamada *aprendizado supervisionado* e o algoritmo de extração de conhecimento é denominado *indutor*. O indutor aceita como entrada um conjunto de exemplos *rotulados*, isto é, um conjunto de exemplos de treinamento no qual cada E_i possui um valor definido y_i , chamado *classe*. A classe é o conceito-meta que relaciona os valores $\mathbf{x}_i$ à conclusão y_i . A premissa da indução sustenta que existe uma função desconhecida f , chamada *função-conceito*, tal que, para cada exemplo $E_i = (\mathbf{x}_i, y_i)$, tem-se $f(\mathbf{x}_i) = y_i$.

O objetivo ideal do aprendizado supervisionado é encontrar ou aproximar essa função conceito f , de modo que a classe pode ser *prevista* pelos valores do vetor $\mathbf{x}_i$ do exemplo. Entretanto, em geral é impossível encontrar f , pois os dados fornecidos ao indutor são geralmente insuficientes para permitir uma caracterização completa da função conceito. Assim, o objetivo prático do indutor é encontrar uma boa aproximação h de f tal que $h(\mathbf{x}) \approx f(\mathbf{x})$. Nesse caso h é uma *hipótese* sobre f . Em outras palavras, do conjunto de possíveis hipóteses H que aproximam f , o indutor deve encontrar uma boa hipótese $h \in H$ (Russell e Norvig, 2003).

Quando o valor de classe y_i pertence a um domínio discreto e limitado (finito) de valores $\{C_1, \dots, C_{N_{Cl}}\}$, a hipótese h é denominada *classificador* e a tarefa de aprendizado pertence ao domínio da *classificação*. Quando o valor de classe y_i pertence a um domínio escalar ou contínuo, a tarefa de classificação é denominada *análise de regressão*. Ainda, se em vez de um único valor, a classe assumir um conjunto de valores – ou rótulos –, trata-se então de classificação ou regressão *multirrótulo*.

No aprendizado não-supervisionado, por sua vez, os exemplos não possuem valor definido y_i . O objetivo do aprendizado não-supervisionado é encontrar um modelo que organize os exemplos por similaridade, podendo ou não ser definido *post-hoc* um conjunto de valores de classe para os exemplos. O exemplo mais recorrente de aprendizado não-supervisionado é o *agrupamento* – ou *clustering* –, no qual exemplos são agrupados em um conjuntos por semelhança de seus valores.

Uma modalidade intermediária entre o aprendizado supervisionado e o aprendizado não-supervisionado é o aprendizado *semisupervisionado*. Nessa modalidade, alguns exemplos do conjunto de dados são rotulados, mas outros não. Esse cenário no qual apenas uma porção do conjunto de dados é rotulada é típico de domínios nos quais deseja-se utilizar técnicas de aprendizado supervisionado, porém a rotulação dos exemplos é complexa ou cara.

2.3.2 Aprendizado Simbólico e Não-Simbólico

Todo algoritmo de AM necessita de um simbolismo formal para representar a hipótese construída a partir dos dados. Com relação a esse aspecto, é possível separar os algoritmos de AM entre algoritmos simbólicos e não-simbólicos, dependendo da característica da linguagem utilizada pelo algoritmo para representar a hipótese. Os termos relacionados aprendizado simbólico e aprendizado não-simbólico remetem à utilização de algoritmos de AM para construir modelos simbólicos e modelos não simbólicos, respectivamente.

Um algoritmo de AM é dito ser *simbólico* quando a hipótese construída por esse algoritmo é representada de forma *simbólica*. Uma representação simbólica da hipótese é aquela que pode ser “lida” por um ser humano com esforços razoáveis. Exemplos de representação simbólica de hipóteses incluem as árvores de decisão e as regras de conhecimento.

Um algoritmo *não-simbólico* representa a hipótese de forma *não-simbólica*. Uma representação não-simbólica é aquela que tipicamente necessita de alguma forma de pós-processamento para ser compreendida por humanos ou não pode ser compreendida a menos que o usuário tenha amplo conhecimento do formato de representação. Um exemplo clássico de representação não-simbólica de conhecimento é a rede neural artificial (Haykin, 1999; Hagan et al., 1996) – RNA. A hipótese contida na RNA é implícita nas interconexões de uma rede de neurônios artificiais, de modo que a interpretação da hipótese é complexa. Outro exemplo é o modelo probabilístico construído pelo classificador *Naïve Bayes* (Zhang, 2004), no qual o conhecimento extraído é representado por uma gama de probabilidades condicionais.

O uso de aprendizado não-simbólico é interessante quando o modelo de conhecimento extraído tem propósito preditivo. Por exemplo, redes neurais artificiais são bastante utilizadas em sistemas de controle de navegação de robôs e veículos automecânicos. Também é bastante comum o uso de sistemas de classificação para procurar transações de crédito fraudulentas, sendo maior a necessidade de que o classificador possa ser aplicado eficientemente a um grande número de exemplos do que a necessidade de se compreender o conhecimento contido nesse classificador. Algoritmos simbólicos, por sua vez, são bastante interessantes quando o pesquisador tem o objetivo de estudar a hipótese extraída dos dados. Ainda que nem sempre os algoritmos indutores sejam capazes de construir hipóteses desconhecidas pelos especialistas do domínio, é possível que os algoritmos indutores forneçam hipóteses que apresentem conhecimento de alguma forma interessante para o especialista, forneçam pistas para auxiliar o próprio especialista a formular hipóteses inéditas ou forneçam evidências empíricas de hipóteses não-comprovadas.

2.4 Terminologia e Linguagem de Descrição

Todo algoritmo de AM necessita de um formalismo para descrever, preferencialmente de forma estruturada, os dados que serão utilizados para a extração do conhecimento, bem como um formalismo de representação do modelo de conhecimento produzido. A representação do conhecimento é de especial importância porque, diferentemente de dados, conhecimento é “um abstrato”. Não se pode esculpir, imprimir, desenhar ou escrever conhecimento, apenas representá-lo através de um modelo que pode ser baseado, por exemplo, em texto, figura ou gráfico. Um modelo de representação do conhecimento é um substituto, uma representação aproximada de algo que não pode existir de forma concreta no “mundo real”.

Sendo um substituto aproximado, um modelo de representação de conhecimento é inexato, e portanto possui limitações. Nem todo conhecimento pode ser representado de forma satisfatória por qualquer modelo de conhecimento. Por exemplo, em um modelo de conhecimento baseado em árvore de decisão possui uma capacidade de representação similar a da lógica proposicional. Sendo assim, esse tipo de modelo não é capaz de representar satisfatoriamente relações entre atributos. Por outro lado, modelos baseados em lógica de primeira ordem, como os tratados em programação lógica indutiva (Nienhuys-Cheng e De Wolf, 1997), podem representar relações entre atributos, mas com custo computacional adicional, uma vez que o espaço de busca pode ser muito maior.

2.4.1 Tabela Atributo-Valor

A *tabela atributo-valor* é uma linguagem de representação de exemplos que apresenta poder de descrição equivalente ao da lógica proposicional. Por sua simplicidade, essa linguagem é adotada pela maioria dos algoritmos de aprendizado.

Na Tabela 2.1 é apresentado o formato geral de uma tabela atributo-valor, composta por um conjunto de N exemplos $E = \{E_1, E_2, \dots, E_N\}$. Cada exemplo $E_i = (\mathbf{x}_i, y_i)$ é composto por um par $(\mathbf{x}_i, y_i)$, sendo cada $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{iM})$ um vetor de valores referente a um conjunto de atributos relacionados $(X_1, X_2, \dots, X_M)$ e y_i um valor de classe relacionado ao atributo-classe Y .

Tabela 2.1: Formato geral da tabela atributo-valor

	X_1	X_2	$\dots$	X_M	Y
E_1	x_{11}	x_{12}	$\ddots$	x_{1M}	y_1
E_2	x_{21}	x_{22}	$\ddots$	x_{2M}	y_2
$\vdots$	$\vdots$	$\ddots$	$\ddots$	$\vdots$	$\vdots$
E_N	x_{N1}	x_{N2}	$\ddots$	x_{NM}	y_N

Na tabela atributo-valor, cada atributo X_j representa uma característica de interesse sobre os exemplos. Um atributo pode ser, por exemplo, a altura de uma pessoa ou a marca de um automóvel, dependendo do domínio relacionado aos dados. Cada elemento x_{ij} da tabela atributo-valor representa o valor que o exemplo E_i assume para o atributo X_j . Dependendo da natureza do atributo, o valor x_{ij} pode ser quantitativo ou qualitativo.

Quantitativo: valores quantitativos podem ser limitados a qualquer conjunto de valores numéricos – *e.g.* reais, inteiros, positivos etc. Quando um atributo pode assumir valores quantitativos, diz-se que o atributo é de natureza quantitativa. É importante observar que, ainda que o domínio do atributo seja limitado, ele deve ser infinito. Caso não seja, trata-se de um atributo que pode ser enumerado e é, portanto, qualitativo;

Qualitativo: valores qualitativos são limitados a um conjunto de valores enumeráveis. Quando um atributo pode assumir valores qualitativos, diz-se que o atributo é de natureza qualitativa. Ocasionalmente, é possível inferir ordem entre os valores – *e.g.* valores inteiros em um intervalo finito –, porém, a maioria dos sistemas de aprendizado não faz distinção entre atributos qualitativos ordenáveis e atributos quantitativos não-ordenáveis;

Em alguns domínios é possível que um ou mais exemplos não apresentem valores definidos para todos os atributos. Isso pode acontecer se houver dependência entre os atributos ou quando não for possível coletar dados completos para todos os exemplos. Por exemplo, se a tabela atributo-valor em questão tiver como exemplos pacientes de um hospital e como atributos exames médicos, é possível que alguns exames sejam custosos demais e não sejam realizados para todos os pacientes, de modo que alguns exemplos conterão atributos não-definidos ou ausentes. Diversos trabalhos, tais como (Batista e Monard, 2003a; Lakshminarayan et al., 1999; Zheng e Low, 1999), lidam com a questão de exemplos com valores ausentes, mas este tema não será abordado com profundidade neste texto.

Como todos os atributos, a classe também pode assumir valores qualitativos ou quantitativos, bem como assumir valores ausentes, conforme discutido na Seção 2.3.1. O caso em que todos os exemplos possuem valores qualitativos refere-se à *indução de classificadores*. O algoritmo de extração de conhecimento é chamado *indutor* e o modelo de conhecimento produzido por esse indutor é chamado *classificador*. É o caso do trabalho descrito neste texto.

Na indução de classificadores através de aprendizado supervisionado, cada exemplo $E_i = (\mathbf{x}_i, y_i)$ possui um valor de classe $y_i \in \{C_1, C_2, \dots, C_{N_{Cl}}\}$. Os valores qualitativos e distintos $C_1, \dots, C_{N_{Cl}}$ são ditos serem as *classes* do conjunto de dados e um exemplo E_i cujo valor de classe é $y_i = C_v$ é dito *pertencer* à classe C_v . O valor inteiro positivo N_{Cl} define a quantidade distinta de valores de classe que os exemplos podem assumir

e é chamado *número de classes*. Um caso típico de indução de classificadores ocorre quando $N_{Cl} = 2$ e é denominado *classificação binária*.

Um exemplo concreto de conjunto de dados representado no formato de tabela atributo-valor é mostrado na Tabela 2.2. O conjunto de dados representado nessa tabela possui 14 exemplos, sendo que cada exemplo possui quatro atributos. O conjunto possui duas classes, com nove exemplos pertencentes à classe Jogar e cinco exemplos pertencente à classe NãoJogar. O domínio do conjunto de dados relaciona um conjunto de atributos climáticos à decisão de um sujeito de ir ou não jogar tênis.

Tabela 2.2: Conjunto de dados tênis representado como tabela atributo-valor – adaptado de (Mitchell, 1997)

	Aspecto	Temperatura	Umidade	Vento	JogarTênis
E_1	Aberto	Alta	Elevada	Fraco	NãoJogar
E_2	Aberto	Alta	Elevada	Forte	NãoJogar
E_3	Nublado	Alta	Elevada	Fraco	Jogar
E_4	Chuvoso	Média	Elevada	Fraco	Jogar
E_5	Chuvoso	Baixa	Normal	Fraco	Jogar
E_6	Chuvoso	Baixa	Normal	Forte	NãoJogar
E_7	Nublado	Baixa	Normal	Forte	Jogar
E_8	Aberto	Média	Elevada	Fraco	NãoJogar
E_9	Aberto	Baixa	Normal	Fraco	Jogar
E_{10}	Chuvoso	Média	Normal	Fraco	Jogar
E_{11}	Aberto	Média	Normal	Forte	Jogar
E_{12}	Nublado	Média	Elevada	Forte	Jogar
E_{13}	Nublado	Alta	Normal	Fraco	Jogar
E_{14}	Chuvoso	Média	Elevada	Forte	NãoJogar

2.4.2 Regras de Conhecimento

Uma das diversas linguagens de representação de conhecimento é a regra de conhecimento. Uma *regra de conhecimento* é um modelo de conhecimento que promove a associação de uma classe $C_v \in \{C_1, C_2, \dots, C_{N_{cl}}\}$ a um exemplo qualquer E_i , condicionada a um conjunto de restrições relacionadas aos atributos do conjunto de dados. Regras de conhecimento são geralmente representadas na forma *R: if complexo then class = C_v* , na qual C_v é uma classe e complexo é uma série de conjunções de termos referentes aos atributos do conjunto de dados. O complexo é também chamado *corpo – body* ou *B* – e a classe associada pela regra ao exemplo é também chamada *cabeça – head* ou *H*.

De forma expandida, a representação geral de uma regra de conhecimento é a seguinte:

$$R : \underbrace{\text{if complexo}}_{\text{body ou } B} \text{ then } \underbrace{\text{class} = C_v}_{\text{head ou } H}$$

Cada termo do corpo da regra segue a forma $X_j \text{ op valor}$, sendo X_j um atributo do domínio dos dados, $\text{op} \in \{<, \leq, \geq, >, =\}$ um operador e valor um valor de restrição. É fundamental que tanto o operador quanto o valor de restrição sejam compatíveis com a natureza do atributo. Por exemplo, um termo na forma $X_j > \xi$ é ilógico se o atributo X_j for de natureza qualitativa e não for possível estabelecer uma ordem para os diferentes valores do domínio desse atributo.

Um exemplo concreto de regra de conhecimento é mostrado na Figura 2.1. O domínio relacionado à regra de conhecimento da Figura 2.1 é o domínio do conjunto *tênis*, apresentado na seção anterior. Apenas exemplos compatíveis com o domínio do conjunto *PlayTennis* podem ser associados à regra.

if Umidade = Normal
then class = Jogar

Figura 2.1: Exemplo de regra de conhecimento relacionada à base de dados *tênis* (apresentada na Tabela 2.2)

A leitura desse regra, com uma certa liberdade, pode ser a seguinte: “se a umidade do ar estiver normal, então o sujeito irá jogar tênis”. O conhecimento contido nessa regra é de que, quando a umidade do ar é normal, então o ambiente é favorável para o jogo. O conhecimento contido em um modelo construído automaticamente sempre pode ser associado a outras informações e conceitos para nos permitir alcançar ideias mais complexas. Por exemplo, com algum conhecimento de climatologia, podemos concluir que, por trás da noção de baixa umidade, está na verdade a perspectiva de que o indivíduo poderá jogar sem ser interrompido por pancadas de chuva. Outro exemplo de regra de conhecimento é mostrado na Figura 2.2, cuja ideia subjacente é facilmente perceptível, apesar de um elemento inédito (*and*) ser apresentado. Isso dá uma ótima noção da alta legibilidade das regras de conhecimento.

if Aspecto = Nublado
and Umidade = Baixa
then class = Jogar

Figura 2.2: Exemplo de regra de conhecimento com conjunção.

A aplicação de uma regra de conhecimento se dá por um teste de cobertura. Dados uma regra $R : B \rightarrow H$ e um exemplo $E_i = (\mathbf{x}_i, y_i)$, testa-se R “contra” E_i verificando se os termos de B são compatíveis com os valores de $\mathbf{x}_i$. Ou seja, para cada termo $X_j \text{ op valor}$ de B verifica-se se a expressão $x_{ij} \text{ op valor}$ é verdadeira. Em caso positivo, diz-se que a regra *cobre* o exemplo e *prediz* que a classe do exemplo é a classe presente em H . Por exemplo, a regra ilustrada na Figura 2.1 cobre os exemplos $E_{5...7}$, $E_{9...11}$ e E_{13} do conjunto de dados *tênis* (Tabela 2.2).

É importante observar que a cobertura não é a única característica importante que pode ser verificada em uma regra de conhecimento. Tipicamente, além de ser capaz de cobrir exemplos, espera-se que regras de conhecimento sejam capazes de prever corretamente os exemplos para serem consideradas úteis. Isto é, espera-se que a

classe predita pela regra seja de fato a classe correta $f(\mathbf{x}_i)$. Além disso, outras características podem ser esperadas. Essa questão será abordada na próxima seção deste texto.

Finalmente, deve-se atentar para o fato de que uma única regra de conhecimento é em geral incapaz de representar totalmente o conhecimento que pode ser extraído de um conjunto de dados, isto é, uma única regra não é capaz de cobrir todos os exemplos ou de apresentar todas as características que se pode esperar de um modelo de conhecimento sobre o domínio. Por essa razão, um classificador raramente é uma única regra de conhecimento, mas um conjunto ou sequência de regras de conhecimento que se complementam com relação à cobertura. Isso é especialmente verdadeiro quando o modelo de conhecimento produzido tem finalidade preditiva. Nessa situação, é interessante que o modelo de conhecimento seja o mais completo possível. Em situações nas quais o processo de KDD é utilizado com finalidade descritiva, entretanto, pode ser mais interessante extrair modelos específicos e inesperados do que modelos capazes de cobrir amplamente o domínio. Nesses casos, a construção de regras de conhecimento individuais pode ser preferível à indução de classificadores compostos por várias regras.

2.4.3 Matriz de Contingência

Nem sempre é suficiente para um usuário verificar que suas regras de conhecimento cobrem um grande número de exemplos. Ou mesmo que suas regras de conhecimento cobrem um grande número de exemplos corretamente. Dependendo da finalidade com a qual o processo de KDD é realizado, cobrir bastante e cobrir bem pode não ser o objetivo principal. A questão latente reside em como avaliar uma regra de conhecimento com relação ao objetivo principal.

Uma forma de se avaliar regras de conhecimento diz respeito a eventos de cobertura e não-cobertura do corpo e da cabeça da regra. O conceito de cobertura do corpo de uma regra $R : B \rightarrow H$ contra um exemplo $E_i = (\mathbf{x}_i, y_i)$ é o conceito de cobertura apresentado na seção anterior. A cabeça da regra, por sua vez, cobre o exemplo quando a classe predita pela cabeça H da regra é a classe verdadeira y_i . Assim, pode-se encarar o teste de uma regra contra um exemplo como a análise de dois eventos, B e H , os quais indicam cobertura do corpo e da cabeça, respectivamente.

Existem apenas dois resultados possíveis para o evento H : ou $H = 1$ e a cabeça da regra cobre o exemplo, ou $H = 0$ e a cabeça não cobre o exemplo. Idem para B . Assim, há quatro cenários distintos para o teste de uma regra de conhecimento contra um exemplo qualquer:

Cobertura da cabeça e do corpo: ocorre quando $H = 1$ e $B = 1$, ou, sinteticamente, HB . É o caso de cobertura correta, isto é, a regra prevê uma classe que pode ser verificada como a classe real do exemplo e representa um conceito correto;

Cobertura da cabeça e não-cobertura do corpo: ocorre quando $H = 1$ e $B = 0$ ou, sinteticamente, $H\bar{B}$. É o caso de regra que representa um conceito correto, mas que não se aplica ao caso (exemplo) em específico;

Não-cobertura da cabeça e cobertura do corpo: ocorre quando $H = 0$ e $B = 1$ ou, sinteticamente, $\bar{H}B$. É o caso de cobertura incorreta, isto é, a regra prevê uma classe que não é a classe real do exemplo e representa um conceito incorreto;

Não-cobertura da cabeça e não-cobertura do corpo: ocorre quando $H = 0$ e $B = 0$ ou, sinteticamente, $\bar{H}\bar{B}$. É o caso de regra que não se aplica ao caso (exemplo) específico e não representa o conceito do caso.

É claro que uma regra de conhecimento R qualquer pode ser testada contra um conjunto de exemplos que não pode ser previsto no momento em que a regra é construída. Isso significa que não se pode definir de antemão quais serão os resultados dos eventos B e H quando a regra for aplicada pelo usuário. Assim, pode-se encarar H e B como duas variáveis aleatórias binárias. Se o conjunto de dados puder ser encarado como uma amostra representativa do domínio no qual o processo de KDD é utilizado, então pode-se estimar a distribuição de probabilidades de B e H para a regra de conhecimento.

A *matriz de contingência*, representada na Tabela 2.3 é uma tabela 2×2 que sintetiza as ocorrências de H e B para uma regra qualquer quando esta é avaliada contra um conjunto de dados. Uma tabela de contingência fornece nove estatísticas para uma regra com relação a um conjunto de dados:

- hb é o número de exemplos nos quais se verifica cobertura da cabeça e do corpo;
- $\bar{h}b$ é o número de exemplos nos quais se verifica apenas cobertura do corpo;
- $h\bar{b}$ é o número de exemplos nos quais se verifica apenas cobertura da cabeça;
- $\bar{h}\bar{b}$ é o número de exemplos nos quais não se verifica cobertura;
- b é o número de exemplos nos quais se verifica cobertura do corpo;
- $\bar{b}$ é o número de exemplos nos quais não se verifica cobertura do corpo;
- h é o número de exemplos nos quais se verifica cobertura da cabeça;
- $\bar{h}$ é o número de exemplos nos quais não se verifica cobertura da cabeça;
- N é o número total de exemplos.

A matriz de contingência também pode ser escrita em termos de frequência relativa, como mostrado na Tabela 2.4. Para se obter a frequência relativa de uma estatística ϵ qualquer, basta encontrar a razão $\frac{\epsilon}{N}$. Neste texto, cada frequência relativa $\frac{\epsilon}{N}$ será representada na forma sintética f_ϵ para fins de simplicidade.

Tabela 2.3: Exemplo de matriz de contingência.

	H	$\overline{H}$	
B	hb	$\overline{h}b$	b
$\overline{B}$	$h\overline{b}$	$\overline{h}\overline{b}$	$\overline{b}$
	h	$\overline{h}$	N

Tabela 2.4: Exemplo de matriz de contingência.

	H	$\overline{H}$	
B	f_{hb}	$f_{\overline{h}b}$	f_b
$\overline{B}$	$f_{h\overline{b}}$	$f_{\overline{h}\overline{b}}$	$f_{\overline{b}}$
	f_h	$f_{\overline{h}}$	1

A matriz de contingência é a base do *framework* de Lavrač (Lavrač et al., 1999). Utilizando os valores absolutos da tabela de contingência ou os valores de frequência relativas, é possível construir muitas medidas de qualidade de regra de conhecimento que estimam a utilidade de uma regra com relação a diferentes propósitos. Algumas dessas medidas serão apresentadas a seguir.

2.5 Qualidade de Regras

Construir regras de conhecimento é uma tarefa relativamente fácil. Em alguns domínios muito particulares, devido ao número favorável de atributos e aos valores factíveis que esses atributos podem assumir, é possível até mesmo enumerar todas as regras de conhecimento factíveis ou um subconjunto significativo delas. Entretanto, é fácil verificar que nem toda regra de conhecimento que pode ser construída possui qualquer utilidade para o usuário. Por exemplo, utilizando o domínio do conjunto *tênis*, duas possíveis regras de conhecimento seriam as regras apresentadas na Figura 2.3.

if Aspecto = Chuvoso
and Vento = Forte
then class = Jogar
 (a)

if Aspecto = Chuvoso
and Temperatura = Alta
and Umidade = Elevada
and Vento = Fraco
then class = Jogar
 (b)

Figura 2.3: Exemplo de regras pouco úteis relacionadas à base de dados *tênis* (apresentada na Tabela 2.2, pág. 33)

Porém, observando a Tabela 2.2, pode-se verificar que nenhum exemplo é coberto corretamente pela regra 2.3 (a), o que significa que a regra não possui qualquer

utilidade preditiva e, provavelmente, qualquer utilidade em absoluto. A regra 2.3 (b), por sua vez, é tão específica ao exemplo E_9 que dificilmente poderá ser utilizada para predição em conjuntos maiores e certamente não apresenta conhecimento mais útil do que aquele que pode ser obtido do próprio exemplo.

É preciso, portanto, que as regras de conhecimento sejam construídas buscando maximizar certos padrões de qualidade. Por exemplo, no caso de modelos preditivos, é de grande interesse que as regras construídas sejam capazes de cobrir corretamente tantos exemplos quanto possível.

Utilizando o *framework* de Lavrač é possível definir inúmeras medidas de qualidade de regras de conhecimento que utilizam as frequências relativas da tabela de contingência para estimar a qualidade da regra segundo diferentes parâmetros. Nesta seção, dois conjuntos de medidas de qualidade de regras são apresentados e discutidos. Na Seção 2.5.1 são apresentadas medidas de qualidade que potencialmente se aplicam melhor a modelos de conhecimento preditivos, ao passo que na Seção 2.5.2 são apresentadas medidas de qualidade que potencialmente se aplicam melhor a modelos de conhecimento descritivos.

2.5.1 Medidas de Qualidade de Regras

Várias medidas têm sido propostas na literatura com o objetivo de avaliar regras de conhecimento segundo vários conceitos de interesse e diferentes abordagens. O trabalho de Lavrač et al. (1999) compila um conjunto de medidas e propõe tratar essas medidas dentro de uma mesmo *framework* utilizando como base a probabilidade de ocorrência dos eventos B e H , os quais podem ser estimados através da matriz de contingência com frequências relativas. A partir do *framework* de Lavrač, pode-se calcular diversas medidas de desempenho, as quais podem estimar a qualidade de uma regra de conhecimento com relação a diferentes parâmetros.

A seguir, são apresentadas 14 medidas de qualidade de regras baseadas na matriz de contingência com frequências relativas. Exceto quando explicitamente especificado diferente, todas as medidas de qualidade resultam em valores no intervalo $[0, 1]$, sendo 0 indicativo de regra de baixa qualidade segundo os critérios da medida e 1 indicativo de regra de alta qualidade.

Precisão: A precisão (consistência ou confiança) é uma medida do quanto uma regra é específica para o problema. A precisão pode ser definida como a probabilidade condicional de H ser verdade dado que B é verdade. Quanto maior, mais precisamente a regra cobre a classe em questão.

$$Pre(R) = \frac{f_{hb}}{f_b} \quad (2.1)$$

Laplace: A medida de precisão não considera o número de exemplos cobertos corretamente, o que privilegia regras que cubram corretamente poucos exemplos. A correção de Laplace tenta contornar esse problema penalizando regras que cubram poucos exemplos. Nessa definição N_{Cl} é o número de classes.

$$LAcc(R) = \frac{f_{hb} + 1}{f_b + N_{Cl}} \quad (2.2)$$

Erro: O erro de uma regra é definido como $1 - Pre(R)$. Quanto maior o erro, menos precisamente a regra cobre a classe em questão e, portanto, menor a qualidade da regra.

$$Err(R) = 1 - Pre(R) = \frac{f_{\overline{h}b}}{f_b} \quad (2.3)$$

Confiança Negativa: é o correspondente à precisão, mas para os exemplos que não são cobertos pela regra. É definida como a probabilidade condicional de H ser falso dado que B também é falso.

$$NegRel(R) = \frac{f_{\overline{h}\overline{b}}}{f_{\overline{b}}} \quad (2.4)$$

Sensitividade: Sensitividade (completude ou *recall*) é uma medida do número (relativo) de exemplos da classe prevista em H cobertos pela regra. É definida como a probabilidade condicional de B ser verdade dado que H é verdade. Quanto maior a sensibilidade, mais exemplos são cobertos pela regra.

$$Sens(R) = \frac{f_{hb}}{f_h} \quad (2.5)$$

Especificidade: é o correspondente à completude, mas para os exemplos que não são cobertos pela regra R . É definida como a probabilidade condicional de B ser falso dado que H é falso.

$$Spec(R) = \frac{f_{\overline{h}\overline{b}}}{f_{\overline{h}}} \quad (2.6)$$

Cobertura: Cobertura é uma medida do número (relativo) de exemplos cobertos pela regra R . É definida como a probabilidade de B ser verdade. Quanto maior a cobertura, maior o número de exemplos cobertos pela regra R .

$$Cov(R) = f_b \quad (2.7)$$

Suporte: Suporte (frequência) é uma medida do número (relativo) de exemplos cobertos corretamente pela regra R . É definido como a probabilidade de H e B serem verdade. Quanto maior o suporte, maior o número de exemplos da classe em questão que são cobertos corretamente pela regra R .

$$Sup(R) = f_{hb} \quad (2.8)$$

Novidade: Novidade pode ser definida como a probabilidade de B e H ocorrerem juntos não poder ser inferida pelas probabilidades de B e H isoladamente, isto é, B e H não são estatisticamente independentes. A medida de novidade é obtida comparando o valor esperado $P(HB)$ com os valores de $P(H)$ e $P(B)$. Quanto mais o valor esperado diferir do observado, maior é a probabilidade que exista uma correlação verdadeira e inesperada entre B e H . Pode ser demonstrado que $-0,25 \leq Nov(R) \leq 0,25$, e quanto maior um valor positivo (mais próximo de 0,25) mais forte é a associação entre B e H enquanto que, quanto maior um valor negativo (mais próximo de -0,25), mais forte é a associação entre B e $\bar{H}$. A medida de novidade é também conhecida como a medida de Piatetsky-Shapiro. Neste trabalho, a medida de novidade é diferenciada da medida de Piatetsky-Shapiro considerando-se que a associação entre B e H é tão boa quanto a associação entre B e $\bar{H}$. Neste trabalho a medida de novidade é também multiplicada por uma constante e assim normalizada para o intervalo $[0, 1]$.

$$Nov(R) = 16 \times (f_{hb} - f_h \times f_b)^2 \quad (2.9)$$

Satisfação: Satisfação é o aumento relativo na precisão entre a regra $B \rightarrow \text{verdade}$ e a regra $B \rightarrow H$. É uma medida mais indicada para tarefas voltadas à descoberta de conhecimento, sendo capaz de promover um equilíbrio entre regras com diferentes condições e conclusões.

$$Sat(R) = \frac{f_h - \frac{f_{hb}}{f_b}}{f_h} \quad (2.10)$$

Precisão Relativa: A precisão relativa de uma regra mede o ganho de precisão obtido em relação à precisão de uma regra padrão $\text{verdade} \rightarrow H$, ou seja, que avalia B como verdade para todos os exemplos. Nesse caso, uma regra só interessa se melhorar a precisão da regra padrão.

$$RAcc(R) = \frac{f_{hb}}{f_b} - f_h \quad (2.11)$$

Confiança Negativa Relativa: É o análogo a precisão relativa para os exemplos que não são cobertos pela regra. Nesse caso, a regra padrão é $\text{falso} \rightarrow \bar{H}$.

$$RNegRel(R) = \frac{f_{h\bar{b}}}{f_{\bar{b}}} - f_{\bar{h}} \quad (2.12)$$

Sensitividade Relativa: A sensibilidade relativa mede o ganho de sensibilidade obtido em relação à sensibilidade de uma regra padrão $B \rightarrow \text{verdade}$, ou seja, uma regra que avalia H como verdade para todos os exemplos.

$$RSens(R) = \frac{f_{hb}}{f_h} - f_b \quad (2.13)$$

Especificidade Relativa: É o análogo a sensibilidade relativa para os exemplos que não são cobertos pela regra. Nesse caso, a regra padrão é $\overline{B} \rightarrow falso$.

$$RSpec(R) = \frac{f_{\overline{h}\overline{b}}}{f_{\overline{h}}} - f_{\overline{b}} \quad (2.14)$$

Um ponto importante referente as medidas relativas (medidas 2.11, 2.12, 2.13 e 2.14) é que elas dão mais informação sobre a utilidade de uma regra que as informações fornecidas pelas suas respectivas medidas absolutas (medidas 2.1, 2.4, 2.5 e 2.6). Por exemplo, se a precisão de uma regra é menor que a frequência relativa da classe que a regra prediz, então a regra tem um desempenho ruim, independentemente de sua precisão absoluta.

Existe, no entanto, um problema com a precisão relativa, pois é fácil obter uma alta precisão relativa para regras muito específicas, ou seja, regras com uma baixa generalidade de $P(B)$. Para contornar esse problema, é proposto em Lavrač et al. (1999) uma variante das medidas relativas, na qual é atribuído um peso para cada uma dessas medidas. Esse peso promove um balanceamento entre a generalidade e a relatividade dessas medidas.

Dessa forma, a medida $RAcc(R)$ é multiplicada pelo seu coeficiente de peso, $P(B)$, obtendo a nova medida **Precisão Relativa com Peso** (2.15), também conhecida na literatura como ganho. É possível mostrar que essa medida é equivalente a $Nov(R)$, ou seja, regras com alta precisão relativa também tem alta novidade e vice-versa.

$$WRAcc(R) = f_b \left(\frac{f_{hb}}{f_b} - f_h \right) \equiv Nov(R) \quad (2.15)$$

Analogamente, as medidas **Confiança Negativa** (2.16), **Sensitividade** (2.17) e **Especificidade** (2.18) **Relativas com Peso** são obtidas multiplicando as correspondentes medidas relativas, $RNegRel(R)$, $RSens(R)$ e $RSpec(R)$, pelos coeficientes de pesos $P(\overline{B})$, $P(H)$ e $P(\overline{H})$, respectivamente. No trabalho de Lavrač et al. (1999) é demonstrado que essas medidas são equivalentes entre si (Equação 2.19), ressaltando a importância da precisão relativa com peso como medida fundamental para a avaliação de regras, promovendo um balanceamento entre precisão e as outras medidas.

$$WRNegRel(R) = f_{\bar{b}} \left(\frac{f_{\bar{h}\bar{b}}}{f_{\bar{b}}} - f_{\bar{h}} \right) \quad (2.16)$$

$$WRSens(R) = f_h \left(\frac{f_{hb}}{f_h} - f_b \right) \quad (2.17)$$

$$WRSpec(R) = f_{\bar{h}} \left(\frac{f_{\bar{h}\bar{b}}}{f_{\bar{h}}} - f_{\bar{b}} \right) \quad (2.18)$$

$$WRAcc(R) \equiv WRNegRel(R) \equiv WRSens(R) \equiv WRSpec(R) \quad (2.19)$$

Muitas outras medidas para avaliação de regras podem ser encontradas na literatura. Por exemplo, em Horst (1999) são descritas outras medidas de qualidade e interesse de regras. Freitas (1998a,b) faz o uso de medidas objetivas para avaliar qualidade e surpresa de regras. Também, o número de nós (atributos) presentes em uma árvores ou o número de condições presentes no complexo de uma regra são usados, normalmente, como medida de compreensibilidade sintática de regras — medida 2.20. Na equação 2.20, $| < complexo > |$ é o número de condições que compõem o complexo e M é o número de atributos presentes no conjunto de dados descrito na tabela atributo-valor.

$$Intel(R) = 1 - \frac{| < complexo > |}{M} \quad (2.20)$$

É importante ressaltar que existe a possibilidade de combinar as medidas aqui apresentadas para auxiliar o usuário no entendimento e utilização do conhecimento adquirido por sistemas de aprendizado. Também vale lembrar que, mesmo que a precisão global do classificador não seja considerada boa, podem existir algumas regras boas em termos de qualidade, novidade, interessabilidade e outras. Esse aspecto é muito importante em KDD (Fayyad et al., 1996a,b; Rezende et al., 2003). Outros trabalhos relacionados à análise do conhecimento de classificadores simbólicos e à combinação de regras de conhecimento utilizando as medidas aqui descritas, foram desenvolvidos por (Gomes, 2002) e (Bernardini, 2006), respectivamente.

2.5.2 Medidas de Interesse

Além das medidas apresentadas na seção anterior, existem medidas de qualidade de regras de conhecimento que focam exclusivamente no problema de extração de conhecimento interessante. A extração de conhecimento interessante é um problema difícil, não havendo um consenso entre os pesquisadores da área sobre qual a forma ideal de resolvê-lo. De fato, em que pese aos pesquisadores da área, não se pode

nem ao menos dizer que existe um consenso quanto ao que pode ser considerado interessante (Geng e Hamilton, 2006).

A despeito disso, diversos trabalhos têm sido realizados com o objetivo de estudar formas de se extrair conhecimento interessante (McGarry, 2005; Freitas, 1998b) e alguns critérios têm sido estabelecidos para se classificar diferentes conceitos de “interesse”. Por exemplo, Geng e Hamilton (2006) argumenta, com base na análise de propostas anteriores, que “[o conceito de] interesse é melhor abordado como um conceito que envolve concisão, generalidade, confiança, peculiaridade, diversidade, novidade, surpresa, utilidade e aplicabilidade”.

Neste trabalho foram pesquisadas algumas medidas de interesse baseadas no critério de generalidade. O critério de *generalidade* envolve a ideia de que uma regra de conhecimento $B \rightarrow H$ é interessante se ela pode ser aplicada ao maior número possível de exemplos que pertence à classe H , cobrindo o menor número possível de exemplos que não pertencem à classe H . Foram levantadas sete medidas de interesse, as quais podem ser explicadas por dois princípios:

Correlação do corpo com a cabeça: a correlação é um conceito estatístico muito importante que envolve a ocorrência conjunta de dois eventos aleatórios. Quando dois eventos são altamente correlacionados, ao verificar-se a ocorrência de um, pode-se esperar com alta probabilidade a ocorrência do outro. Além disso, caso seja verificada a não-ocorrência de um evento, pode-se esperar com alta probabilidade a não-ocorrência do outro.

É muito importante observar que correlação não é sinônimo de causalidade, isto é, a correlação entre dois eventos não significa que um possa ser explicado como a causa do outro. Apesar disso, a existência de correlação é um indicativo de que pode haver causalidade ou que pode haver uma causa em comum entre os dois eventos. Por exemplo, a observação de estrelas no céu é um evento altamente correlacionado com o por-do-sol. Quando o sol se põe, pode-se esperar com alta probabilidade que seja possível observar estrelas no céu e, se observamos estrelas no céu, pode-se esperar com alta probabilidade que o sol tenha se posto. Entretanto, não se pode dizer que o por-do-sol assegura que estrelas serão observadas – por exemplo, o céu pode estar nublado –, tampouco que o por-do-sol é a causa pela qual estrelas podem ser observadas ou mesmo que o sol tenha necessariamente se posto quando estrelas são observadas no céu. Ainda assim essa correlação sugere que ambos os fenômenos, se não podem ser explicados um como causa do outro, possuem alguma explicação causal em comum. A verificação dessa correlação auxilia a compreensão dos fenômenos envolvidos.

Dada uma regra de conhecimento $B \rightarrow H$ qualquer, diz-se que existe *correlação positiva* entre o corpo e a cabeça da regra se B ocorre com alta probabilidade quando H ocorre e vice-versa. Uma regra que apresente alta correlação positiva cobre muito bem os exemplos da classe H “evitando” cobrir exemplos das demais

classes. Em outras palavras, uma regra com alta correlação entre B e H atende ao critério de generalidade e pode ser interessante.

Por outro lado, diz-se que existe *correlação negativa* entre o corpo e a cabeça da regra se B ocorre com alta probabilidade quando H não ocorre e H ocorre com alta probabilidade quando B não ocorre. Uma regra com alta correlação negativa pode ser interessante ou não, dependendo do domínio de aplicação. Por exemplo, no caso de classificação binária, uma regra com alta correlação negativa pode ser facilmente transformada em uma regra com alta correlação positiva alternando-se a classe predita pela regra. No caso em que o número de classes é superior a dois, uma regra com alta correlação negativa pode ser interessante para estabelecer casos nos quais o conceito da classe H não é verdadeiro.

Finalmente, se a correlação entre B e H for nula, então a regra certamente não apresenta uma relação de causa ou de causa comum entre B e H , o que significa que a regra muito provavelmente não é interessante. Observe que, por questões de simplicidade, regras com correlação negativa, independente do grau de correlação, são consideradas tão desinteressantes quanto regras com correlação nula.

Precisão vs. distribuição da classe: algumas medidas de interesse são baseadas em uma relação que pode ser verificada entre a precisão de uma regra $B \rightarrow H$ e a frequência da classe H . Se a precisão da regra for inferior à frequência da classe, então é esperado que o conhecimento expresso pela regra tenha pouca utilidade preditiva, pois sua precisão é inferior ao de uma regra padrão que sempre prevê a classe H .

Por exemplo, se uma regra de conhecimento $B \rightarrow H$ tem precisão estimada de 0.75, então ela é capaz de prever corretamente cerca de 75% dos exemplos cobertos. Entretanto, caso essa regra seja aplicada em um domínio desbalanceado, no qual a proporção de exemplos da classe H corresponde a, diga-se, 80%, então seria mais produtivo classificar todos os exemplos como pertencentes à classe H e, assim, teria-se uma taxa preditiva de 80%.

Regras que sejam condizente com esse princípio satisfazem $\frac{f_{hb}}{f_b} > f_h$. Medidas de conhecimento que podem ser explicadas por esse princípio tipicamente envolvem expressões como $f_{hb} - f_h \times f_b$, a exemplo da medida de Piatetsky-Shapiro. Essa medida assume o valor zero quando a regra possui precisão idêntica à frequência da classe, sendo positiva quando a precisão é superior à frequência da classe e negativa no caso oposto.

Observe que o conceito de precisão vs. distribuição da classe é semelhante ao conceito de correlação. Quanto maior a razão entre a precisão da regra e a frequência da classe, maior é a cobertura de exemplos corretos e menor a cobertura de exemplos incorretos. Em outras palavras, maior é a correlação entre B e H . O raciocínio inverso também é válido. Apesar disso, os dois conceitos são utilizados para explicar as medidas de conhecimento porque algumas são me-

lhora compreendidas em sob a ótica da correlação, enquanto outras são melhor compreendidas sob a ótica da relação entre precisão e frequência da classe.

Como no caso da correlação, os casos nos quais a precisão da regra é inferior à frequência da classe são considerados tão desinteressantes quando a precisão da regra é idêntica à frequência da classe.

A seguir são apresentadas sete medidas de interesse extraídas da literatura. A maioria dessas medidas apresentadas foi inicialmente proposta para estimar o interesse de regras de associação na forma $R : X \rightarrow Y$. Regras de associação indicam a ocorrência de dois conjuntos de itens, ou *itemsets*, X e Y . Apesar dessa distinção, existe uma certa semelhança entre regras de associação e regras de conhecimento para classificação e as ocorrências dos eventos B e H em conjuntos de dados estatisticamente relevantes é equivalente à ocorrência dos *itemsets* X e Y .

Φ -Coefficient: Agresti (2002) propõe uma medida de interesse baseada na correlação entre duas variáveis aleatórias binárias, semelhante ao coeficiente de correlação de Pearson. A medida *Phi* utilizada neste trabalho e descrita pela Equação 2.21 é uma variação da medida *Phi* que envolve o conceito de precisão e distribuição de classe (Tan et al., 2002).

$$Phi(R) = \frac{f_{hb} - f_h \times f_b}{\sqrt{f_h \times f_b \times f_{\bar{h}} \times f_{\bar{b}}}} \quad (2.21)$$

Odds Ratio: Dado um evento ϵ qualquer, a *chance* de ocorrência desse evento é definida como sua probabilidade de ocorrência sobre sua probabilidade de não-ocorrência. Por exemplo, a chance de se obter o resultado 3 da rolagem de um dado comum e honesto é 1 : 5 – ou “uma para cinco”. Em se tratando de regras de conhecimento, assumindo-se que ocorre cobertura da cabeça, a chance de haver cobertura do corpo é $f_{hb} : f_{h\bar{b}}$. Por outro lado, assumindo-se que não ocorre cobertura da cabeça, a chance de haver cobertura do corpo é $f_{\bar{h}b} : f_{\bar{h}\bar{b}}$.

A medida de *Odds Ratio* (Mosteller, 1968) é definida como a razão entre as chances de haver cobertura do corpo quando ocorre e quando não ocorre cobertura da cabeça. Dessa forma, a medida estima uma razão entre a proporção de exemplos que a regra cobre corretamente e a proporção de exemplos que a regra cobre incorretamente. Dadas regras com valores próximos de cobertura, aquela que apresenta maior valor de *Odds Ratio* tende a cobrir melhor os exemplos da classe que prevê, sem cobrir exemplos da outra classe, o que pode significar que o conhecimento representado pela regra generaliza bem o conceito da classe.

$$Odds(R) = \frac{f_{hb} \times f_{\bar{h}\bar{b}}}{f_{h\bar{b}} \times f_{\bar{h}b}} \quad (2.22)$$

Kappa: A medida Kappa ou K (Cohen, 1960) avalia o grau de concordância entre duas variáveis. Se houver uma alta concordância entre a ocorrência do corpo e

da cabeça da regra, o valor da medida tende a ser elevado – $Kap \rightarrow 1$. Se houver uma concordância entre a ocorrência do corpo e de um valor distinto da cabeça da regra, o valor da medida tende a ser reduzido – $Kap \rightarrow -1$. Caso não haja correlação entre B e H ou entre B e $\overline{H}$, o valor da medida tende a zero.

$$K(R) = \frac{f_{hb} + f_{hb} - f_h \times f_b - f_{\overline{h}} \times f_{\overline{b}}}{1 - f_h \times f_b - f_{\overline{h}} \times f_{\overline{b}}} \quad (2.23)$$

Interest: o fator de interesse ou I (Brin et al., 1997; Silverstein et al., 1998) é uma medida utilizada com bastante frequência em *data mining* na construção de regras de associação. A medida é baseada em um cálculo de dependência estatística entre os conjunto de itens associados pelas regras e, quando aplicada a regras de conhecimento, percebe-se que também embute claramente a ideia de compromisso entre precisão e cobertura. Como $f_{hb}, f_h, f_b < 0$, se a regra possui precisão superior à frequência da classe, então f_{hb} tende a ser maior que $f_h \times f_b$ e a medida tende ao infinito. Quando a precisão da regra é exatamente a frequência da classe, tem-se $I(R) = 1$. Por outro lado, à medida que a precisão da regra torna-se inferior à frequência da classe, o valor de $I(R)$ tende a zero.

$$I(R) = \frac{f_{hb}}{f_h \times f_b} \quad (2.24)$$

Cosine: a medida *Cosine* – IS – é proposta por Tan e Kumar (2000) com base na medida I . A medida IS é tipicamente interpretada como o cosseno de dois vetores em modelos de aprendizado vetoriais, porém também pode ser vista como a média geométrica entre o coeficiente de interesse e o suporte da regra.

$$IS(R) = \frac{f_{hb}}{\sqrt{f_h \times f_b}} \quad (2.25)$$

Piatetsky-Shapiro's: a medida de Piatetsky-Shapiro ou PS (Piatetsky-Shapiro, 1991) é normalmente tida como um exemplo clássico de medida baseada na correlação de duas variáveis. Essa interpretação advém do fato que a probabilidade conjunta de dois eventos B e H ocorrerem simultaneamente é $P(B, H) = P(B) \times P(H)$ se estes forem independentes. Por outro lado, quando há correlação máxima entre B e H , observa-se que $P(B|H) = P(H|B) = 1$ e também que $P(B, H) = P(B) = P(H)$. Assim, quando não há correlação $PS(R) = 0$, ao passo que $PS(R) = 0.25$ quando a correlação é máxima.

Por outro lado, a medida de Piatetsky-Shapiro também pode ser interpretada como uma aplicação direta do conceito de precisão e distribuição da classe – o que não deve surpreender, já que ambos os conceitos se relacionam ao mesmo princípio de interesse. Se a regra possui cobertura superior à frequência da classe, então $f_{hb} > f_h \times f_b$ e $PS(R)$ tende a ser positivo. Se a regra possui cobertura inferior, a desigualdade se inverte e $PS(R)$ tende ao negativo. No caso em que a regra possui cobertura idêntica à frequência da classe, $PS(R) = 0$.

$$PS(R) = f_{hb} - f_h \times f_b \quad (2.26)$$

Jaccard: o coeficiente de Jaccard (Rijsbergen, 1979) é um estimador estatístico utilizado extensamente na literatura para avaliar o grau de similaridade entre conjuntos de itens. Esta medida é melhor compreendida quando se considera que a probabilidade de ocorrência de pelo menos um dentre dois eventos B e H é $P(B \cup H) = P(B) + P(H)$ se B e H forem independentes e $P(B \cup H) = P(B) + P(H) - P(B, H)$ se houver dependência entre eles. Assim o coeficiente de Jaccard tende a 1 quanto maior for a correlação entre B e H . Caso contrário a medida tende a zero conforme o suporte da regra diminui – observe que $f_h + f_b > f_{hb}$.

$$Jac(R) = \frac{f_{hb}}{f_h + f_b - f_{hb}} \quad (2.27)$$

Observe que nem todas as medidas de interesse apresentadas nesta seção conseguem captar perfeitamente a noção de correlação entre B e H . Algumas, como o coeficiente de interesse, não têm utilidade quando a regra não cobre nenhum exemplo ou quando a regra não cobre nenhum exemplo da classe que prediz – $I(R)$ tende não tem limite definido nessa situação.

Isso não invalida as medidas ou justifica seu descarte para este trabalho. Como essas medidas são propostas para a avaliação de regras de associação, as quais são tipicamente mineradas com parâmetros mínimos de cobertura e suporte, é rara a necessidade de utilizá-las em contextos nos quais a regra não cobre nenhum exemplo ou não cobre corretamente nenhum exemplo. Neste trabalho, por sua vez, as medidas podem ser utilizadas em conjunto com outras medidas, como cobertura, precisão e suporte, sendo necessário apenas a aplicação de mecanismos para evitar que essas medidas sejam utilizadas em contextos nos quais envolveriam operações inválidas – *e.g.* divisão por zero.

Na Tabela 2.5 é apresentado um sumário das medidas de interesse, incluindo intervalos de domínio e de interesse, os quais não foram apresentados anteriormente. Os intervalos de interesse são sub-intervalos dos intervalos de domínio nos quais as regras são comparáveis de acordo com a medida. As regras são consideradas incomparáveis para valores fora dos intervalos de interesse porque estes indicam correlação negativa ou precisão inferior à frequência da classe.

2.6 Considerações Finais

Neste capítulo foram introduzidos alguns conceitos importantes para a realização deste trabalho. Em especial, foi descrita a matriz de contingência para regras e o *framework* de Lavrač para a descrição de medidas de qualidade de regras.

Tabela 2.5: Sumário das medidas de interesse. *Obs.*⁽¹⁾: quando a precisão da regra é idêntica à frequência da classe, a medida Cosseno equivale a $\frac{f_{hb}}{\sqrt{f_{hb}}} = \sqrt{f_{hb}}$. *Obs.*⁽²⁾: o coeficiente de Jaccard não possui intervalo indicativo de correlação negativa.

Medida	Equação	Intervalo	Intervalo de Interesse
Φ	$\frac{f_{hb} - f_h \times f_b}{\sqrt{f_h \times f_b \times f_{\bar{h}} \times f_{\bar{b}}}}$	$(-\infty, +\infty)$	$[0, +\infty)$
$Odds$	$\frac{f_{hb} \times f_{\bar{h}\bar{b}}}{f_{h\bar{b}} \times f_{\bar{h}b}}$	$[0, +\infty)$	$[1, +\infty)$
Kap	$\frac{f_{\bar{h}\bar{b}} + f_{hb} - f_h \times f_b - f_{\bar{h}} \times f_{\bar{b}}}{1 - f_h \times f_b - f_{\bar{h}} \times f_{\bar{b}}}$	$[-1, +1]$	$[0, +1]$
I	$\frac{f_{hb}}{f_h \times f_b}$	$[0, +\infty)$	$[1, +\infty)$
IS	$\frac{f_{hb}}{\sqrt{f_h \times f_b}}$	$[0, +\infty)$	$[\sqrt{f_{hb}}, +\infty)^{(1)}$
PS	$f_{hb} - f_h \times f_b$	$[-0.25, +0.25]$	$[0, +0.25]$
Jac	$\frac{f_{hb}}{f_h + f_b - f_{hb}}$	$[0, 1]$	$[0, 1]^{(2)}$

Tais medidas são importantes para esta Dissertação, pois podem ser utilizadas para representar os critérios que o usuário deseja que sejam utilizados no processo de extração de conhecimento. Apesar disso, deve-se reiterar que os critérios à disposição do usuário não se limitam às medidas apresentadas neste capítulo, podendo o usuário utilizar quaisquer medidas que possam ser descritas segundo o *framework* de Lavrač ou com base em outras características das regras. Isso difere de outros trabalhos de indução de regras na literatura, os quais geralmente realizam uma busca por regras que maximizem cobertura e precisão ou alguma medida relacionada aos conceitos de generalidade e corretude.

Como mencionado anteriormente, a abordagem para indução de regras com propriedades desejadas utilizada neste trabalho consiste em utilizar computação evolutiva multiobjetivo para otimizar medidas de qualidade de regras como objetivos de um problema de otimização multiobjetivo. Por esse motivo, nos Capítulos 3 e 4 é realizada uma introdução à otimização a aos algoritmos evolutivos, respectivamente.

Otimização

3.1 Considerações Iniciais

Otimização é um processo que, dado um conjunto de objetivos e restrições, visa encontrar uma ou mais soluções que maximizem (ou minimizem) os objetivos e, ao mesmo tempo, respeitem as restrições impostas. Problemas de otimização existem nos mais variados domínios, sendo frequentemente resultantes do mapeamento de uma situação do mundo real para um modelo matemático. Por exemplo, uma fábrica pode empregar otimização para descobrir a melhor estratégia de produção e venda buscando obter lucro máximo. Ou então uma equipe de administradores pode empregar otimização para decidir a melhor forma de aplicar recursos.

Otimização pode ser utilizada em aprendizado de máquina como uma forma de construir modelos que otimizem um ou mais aspectos desejáveis desses modelos. Por exemplo, pode-se utilizar otimização para encontrar regras de conhecimento tal que a cobertura e a precisão sejam máximas. Ou pode-se utilizar otimização para encontrar regras de conhecimento com máximo suporte e uma medida de interesse qualquer.

Neste capítulo são introduzidos os conceitos de otimização utilizados na execução deste trabalho. Na Seção 3.2 são descritos o problema geral de otimização e os conceitos fundamentais. Na Seção 3.3 é feita uma distinção entre problemas de otimização simples-objetivo e problemas de otimização multiobjetivo, os quais são de interesse deste trabalho. Finalmente, na Seção 3.4 são explicadas algumas técnicas utilizadas para avaliação de soluções de um problema multiobjetivo utilizando composição de *rankings*.

3.2 Otimização Simples-Objetivo

No caso mais simples, o problema de otimização é modelado como um conjunto de variáveis de decisão e uma função objetivo. As *variáveis de decisão* são parâmetros

relacionados ao problema em questão – *e.g.* número de peças fabricadas ou montante investido em um fundo. A *função objetivo* é uma função que associa as variáveis de decisão a um valor real. Uma *solução* é um vetor de valores associados às variáveis de decisão que descreve uma possível estratégia de resolução do problema de otimização, sendo uma *solução ótima* aquela para a qual a função objetivo assume o máximo ou mínimo valor possível, dependendo do problema em questão. Soluções que não são ótimas são chamadas *sub-ótimas*. Um exemplo de modelo para esse tipo de problema de otimização é dado na Equação 3.1.

$$\begin{aligned} (\min) \quad & f(x, y) = x^2 + y^2 \\ & x, y \in \mathbb{R} \\ & f : \mathbb{R}^2 \rightarrow \mathbb{R} \end{aligned} \tag{3.1}$$

A leitura desse modelo é a seguinte: deseja-se encontrar um par de valores reais (x, y) tal que o valor da função f seja o menor possível para esse par. Em outras palavras, deseja-se *minimizar* a função f .

Duas possíveis soluções para o problema modelado através da Equação 3.1 são, por exemplo, $S_1 = (2, 2)$ e $S_2 = (0, 0)$. Observe que $f(S_1) = 4$ e $f(S_2) = 0$. Como trata-se de um problema de minimização, então a solução S_2 é melhor – ou mais eficiente ou de melhor qualidade – que a solução S_1 e, conseqüentemente, pode-se afirmar que a solução S_1 é sub-ótima. É possível verificar também que não existe solução mais eficiente que S_2 , o que significa que S_2 é a solução ótima, como pode ser facilmente observado na Figura 3.1.

Figura 3.1: A figura retrata o espaço de decisão do problema de otimização modelado na Equação 3.1. As coordenadas x e y indicam valores das variáveis de decisão. A terceira coordenada indica o valor assumido pela função objetivo quando as variáveis de decisão assumem os valores relacionados. As linhas vermelhas contínuas na base da figura indicam as coordenadas e a “altura” da solução ótima.

Veja que, para o problema apresentado, qualquer par de valores reais (x, y) representa uma solução, distinguindo-se os pares apenas na qualidade das soluções associadas.

Isso ocorre porque esse problema de otimização não possui *restrições*, isto é, limites aos parâmetros do problema. Problemas reais raramente podem ser modelados sem a inclusão de restrições, as quais são tipicamente modeladas como desigualdades envolvendo as variáveis de decisão, pois é incomum que cenários do mundo real sejam irrestritos. Uma fábrica, por exemplo, não pode produzir uma quantidade ilimitada de produtos, assim como uma equipe de administradores provavelmente possui cotas mínimas de investimentos em certas áreas, não sendo possível aplicar todos os recursos em um único investimento rentável.

Para ilustrar essa questão, imagine um indivíduo que está realizando preparativos para viajar como “mochileiro”. Ele tem à disposição um conjunto de N itens que podem ser levados na viagem, sendo que cada item i possui um valor de importância p_i e um peso w_i . O viajante gostaria de levar consigo tantos itens distintos quanto possível, porém ele está restrito a um subconjunto de itens cujo peso total seja inferior a w_{max} , que é a capacidade da mochila. O objetivo do viajante é, portanto, encontrar um subconjunto de itens de forma a maximizar a importância dos itens sem que o peso total exceda a capacidade da mochila. Tal problema poderia ser modelado, com restrições, de forma semelhante ao apresentado na Equação 3.2 a seguir.

$$\begin{aligned}
 (max) \quad & f(\mathbf{x}) = p_1 \times x_1 + p_2 \times x_2 + \dots + p_N \times x_N \\
 & \mathbf{x} = (x_1, x_2, \dots, x_N) \\
 & w_1 \times x_1 + w_2 \times x_2 + \dots + w_N \times x_N < w_{max} \\
 & x_i = 1, \text{ se o viajante opta por levar o item } i \\
 & x_i = 0, \text{ se o viajante opta por não levar o item } i \\
 & p_i \in \mathbb{R} \quad \forall i = 1 \dots N \\
 & w_i \in \mathbb{R} \quad \forall i = 1 \dots N \\
 & f : \{0, 1\}^N \rightarrow \mathbb{R}
 \end{aligned} \tag{3.2}$$

Esse problema é bastante recorrente e é conhecido como o problema da mochila binária. Muitos problemas de otimização podem ser reduzidos ao problema da mochila binária ou a variantes do problema da mochila binária. Como esses problemas são bastante estudados, reduzir um problema de otimização qualquer ao problema da mochila binária ou a uma variante permite resolver o problema de otimização com o uso de técnicas bem-estabelecidas.

Uma distinção importante do problema da mochila apresentado na Equação 3.2 para o problema de otimização anteriormente apresentado diz respeito à factibilidade das soluções. Enquanto no primeiro problema qualquer par (x, y) constitui uma solução possível, nem toda combinação de itens pode ser utilizada para solucionar o problema da mochila. Considere, por exemplo, que o viajante precise escolher entre os itens apresentados na Tabela 3.1.

Uma possível solução para o problema seria o conjunto de itens composto pelo saco de dormir, pelo travesseiro e pelo dinheiro, o qual oferece a importância total 12 e

Tabela 3.1: Exemplo de itens relevantes para o problema da mochila.

Item	Importância	Peso
Saco de dormir	4	5
Travesseiro	3	1
Comida	1	3
Lanterna	1	2
Dinheiro	5	1

impõe o peso total 7. Esta seria uma solução ótima para o problema caso a mochila tivesse capacidade 7. Entretanto, se a mochila tivesse capacidade 6 ou inferior, essa solução excederia a capacidade máxima permitida e, portanto, não seria factível.

Soluções factíveis são aquelas cujos valores associados às variáveis de decisão respeitam as restrições do problema. Por exemplo, considerando-se o problema da mochila de capacidade 6 com os itens exemplificados na Tabela 3.1, uma solução factível seria o conjunto de itens composto pelo saco de dormir e pelo dinheiro, o qual oferece a importância total 9 e impõe o peso total 6. Outra solução factível seria o conjunto de itens composto pelo travesseiro, pelo dinheiro e pela lanterna, também oferecendo importância total 9 e impondo peso total 4. Como o problema de otimização considera apenas o interesse total da mochila, não havendo preferência por mochilas mais leves, ambas as soluções são igualmente boas. Adicionalmente, é possível verificar que ambas são soluções ótimas.

A observação de que não existe preferência por mochilas mais leves é importante porque permite classificar esse problema de otimização como um problema simples-objetivo. Problemas de *otimização simples-objetivos* são aqueles que podem ser modelados com uma única função objetivo. É comum que tais problemas possuam uma única solução ótima, porém, em alguns casos, diversas soluções podem resolver o problema de otimização com igual qualidade. Tais problemas são ditos *multimodais*. Quando se procura encontrar todas as soluções ótimas – ou um subconjunto razoável delas – de um problema multimodal, diz-se estar realizando *otimização multimodal*. Otimização multimodal de problemas simples-objetivos é aplicada quando o usuário deseja ser capaz de escolher uma dentre várias soluções ótimas com base em critérios que não foram definidos como objetivos ou restrições do problema.

Perceba que essa escolha com base em critérios inéditos remete a um interesse subjetivo do usuário. Pode ser preferível utilizar otimização multimodal a descrever esse interesse subjetivo como parte do problema de otimização porque essa descrição pode ser difícil de se obter ou porque o usuário não pode prever com segurança os critérios que ele desejará utilizar no momento de realizar a escolha final da solução. Por exemplo, considerando-se o problema da mochila, o usuário pode estar preocupado com a interação entre os itens da mochila ou pode não ser capaz de verificar que um item qualquer é essencial até ter examinado diferentes configurações de mochilas com importância máxima.

Por outro lado, existem situações nas quais o usuário é capaz de descrever mais de um objetivo para o problema de otimização, sejam esses objetivos interesses subjetivos ou não. Tais casos podem constituir problemas de otimização multiobjetivo, os quais demandam técnicas e conceitos distintos daqueles empregados em otimização simples-objetivo. Algumas dessas técnicas e os conceitos serão abordados nas próximas seções deste capítulo.

3.3 Otimização Multiobjetivo

Problemas de otimização simples-objetivo envolvem a maximização ou minimização de uma única função. Entretanto, não raro problemas do mundo real não podem ser modelados com uma única função objetivo. Frequentemente, existem vários objetivos envolvidos em um processo de tomada de decisão e, muitas vezes, esses objetivos são conflitantes entre si. Por exemplo, o problema de investir em ações pode ser modelado como um problema de otimização que busca, simultaneamente, maximizar o retorno e minimizar o risco. Entretanto, é comum que investimentos de maior retorno sejam também investimentos de maior risco, de modo que ao buscar soluções com maior retorno encontra-se também soluções com maior risco – e a contra-positiva se aplica. Os objetivos retorno e risco são, portanto, conflitantes. Problemas com diversas funções objetivo e pelo menos duas funções conflitantes são conhecidos como problemas de *otimização multiobjetivo*.

Problemas de otimização multiobjetivo exigem abordagens e conceitos distintos daqueles aplicados na análise e resolução de problemas simples-objetivo. Mesmo o conceito de “ótimo” é diferente neste caso, pois é preferível obter um equilíbrio entre todos os objetivos a otimizar apenas alguns poucos (Coello, 2006). Considere, por exemplo, o problema que envolve a compra de um carro cujo conforto seja máximo e cujo custo seja mínimo, ilustrado na Figura 3.2.

Figura 3.2: Exemplo de espaço de decisão multiobjetivo (Deb, 2001)

Sete soluções são indicadas na figura pelos pontos A, B, ... e G, representando sete carros distintos. As características dos carros, que seriam modeladas como as va-

riáveis de decisão do problema, não são mostradas. Em vez disso, a figura ilustra o *espaço de objetivos*, indicando como cada solução otimiza os dois objetivos: conforto (maximização) e custo (minimização). Não existe uma única solução ótima que permita uma escolha fácil, maximizando o conforto e ao mesmo tempo minimizando o custo, tampouco existem diversas soluções que maximizem ambos os objetivos de forma idêntica. Porém, com base no espaço de objetivos, é fácil comparar as soluções e verificar que algumas são sub-ótimas. Por exemplo, a solução E fornece custo R\$100 mil e conforto 90%, ao passo que a solução G fornece custo R\$100 mil e conforto 70%. Como ambas as soluções têm mesma qualidade com relação a um objetivo (custo), aquela que é inferior com relação ao segundo objetivo (conforto) é de modo geral inferior e, portanto, sub-ótima. De forma semelhante, a existência da solução B permite concluir que a solução F é sub-ótima.

As soluções F e G podem ser consideradas sub-ótimas porque são dominadas pelas soluções E e B. Diz-se que uma solução é *dominada* por outra solução quando ela é inferior à outra com relação a pelo menos um objetivo e não é superior com relação a nenhum objetivo. Dado um conjunto de soluções quaisquer um problema multiobjetivo, é possível separar as soluções entre dominadas e não-dominadas. O conjunto de soluções não-dominadas é chamado *fronteira de Pareto*, devido à relação com o conceito de eficiência de Pareto proposto pelo economista italiano Vilfredo Pareto (1848 – 1923) e ao fato de sempre formar uma “fronteira” no espaço de objetivos, como pode ser observado na Figura 3.2 – os pontos de A a E indicam os valores que cinco soluções oferecem às funções objetivo e a linha tracejada é uma interpolação desses pontos, indicando as possíveis localizações de outras soluções não-dominadas. Observe que a fronteira de Pareto pode ser particular não apenas ao problema de otimização multiobjetivo, mas também ao conjunto de soluções que figuram nesse conjunto. Um conjunto diferente de soluções poderia ser encontrado para o mesmo problema, possivelmente fornecendo uma fronteira de Pareto distinta. Se a fronteira de Pareto é tal que nenhuma solução factível pode dominar alguma dessas soluções, esta é chamada *fronteira Pareto-ótima*. Exemplos de fronteiras são mostrados na Figura 3.3.

A resolução de problemas de otimização multiobjetivo tipicamente consiste em progressivamente buscar soluções que permitam detectar e descartar aquelas que são dominadas. Idealmente, em algum momento o conjunto de soluções obtidas será tal que nenhuma solução pertencente a esse conjunto pode ser dominada por qualquer solução factível. Entretanto, em se tratando de otimização multiobjetivo, não é possível decidir entre soluções não-dominadas sem acrescentar novos critérios, o que significa que um sistema computacional dificilmente pode fornecer a resposta ideal para uma situação multiobjetivo sem o auxílio de um humano. Por isso, um conjunto de soluções sub-ótimas, porém variadas e “razoavelmente” próximas da fronteira Pareto-ótima – Figura 3.3 (a) — pode ser frequentemente preferível a um conjunto de soluções ótimas, porém muito semelhantes – Figura 3.3 (b).

Figura 3.3: Duas fronteiras de Pareto para um problema de otimização (max,max). A linha contínua indica a fronteira Pareto-ótima. Os pontos representam diferentes soluções no espaço de objetivos. A linha tracejada é uma interpolação da fronteira de Pareto que contém as soluções ilustradas.

Diversos métodos de resolução de problemas de otimização multiobjetivo buscam encontrar soluções o mais próximas possível da fronteira Pareto-ótima e, ao mesmo tempo, oferecendo boa diversidade no atendimento dos objetivos. Nos capítulos seguintes deste texto serão abordadas técnicas de computação evolutiva multiobjetivo e a biblioteca de classes ECLE, desenvolvida com o propósito de construir regras de conhecimento por meio de otimização multiobjetivo.

3.4 *Rankings* para Otimização Multiobjetivo

As técnicas de otimização multiobjetivo podem ser separadas em dois grupos: aquelas que utilizam o conceito de dominância para encontrar um conjunto de soluções não-dominadas e aquelas que convertem o problema multiobjetivo em um problema simples-objetivo. Essa conversão permite a utilização de técnicas simples-objetivo na resolução de problemas multiobjetivo. Porém, tal conversão não é “perfeita”, acarretando alguns problemas que precisam ser tratados, conforme será discutido em breve.

A forma mais simples de se converter um problema multiobjetivo em um problema simples-objetivo é compor diversas funções objetivos por meio de uma média aritmética ponderada. Assim, um problema de otimização que visa maximizar ou minimizar um conjunto de funções $f_i(\mathbf{x})$ $i = 1 \dots M$ pode ser composto em uma única função $F(\mathbf{x}) = \sum_{i=1}^M \frac{w_i \times f_i(\mathbf{x})}{M}$. Entretanto, a definição de pesos é em geral difícil, podendo em alguns casos ser impossível. Pois tal composição pode envolver a combinação de valores numéricos que pertencem a escalas distintas ou de dimensões incompatíveis. Por exemplo, suponha um processo de otimização que vise simultaneamente maxi-

mizar a resistência de uma estrutura metálica, minimizar o custo de suas partes e minimizar o tempo de construção de cada estrutura individual na linha de montagem. Tais objetivos, além de potencialmente pertencer a escalas muito distintas, envolvem conceitos representados por dimensões incompatíveis. É provável que até mesmo para um especialista da área seja complicado encontrar uma forma lógica de se combinar aritmeticamente funções envolvendo pressão (N/m^2), preço ($R\$$) e tempo (s), pois não existe nenhum conceito que seja normalmente representado em alguma dimensão envolvendo as três.

A menos que sejam definidos pesos condizentes para as funções, as disparidades de dimensionalidade e de escala tornam a composição de funções por média aritmética altamente questionável. Uma forma de contornar essa situação consiste em utilizar *rankings* na composição da função F . Com isso todos os objetivos passam a ser representados por um fator adimensional que representa o mesmo conceito, *i.e.*, o desempenho de cada solução com relação às demais.

Um *ranking* nada mais é do que uma ordem parcial em um conjunto de elementos. Por exemplo, ao se ordenar um grupo de pessoas por altura, tem-se um *ranking* segundo o critério de altura. A posição de cada indivíduo no *ranking* é chamada *rank*¹. É comum, mas não obrigatório, denotar os *ranks* por números ordinários 1, 2, ..., n , sendo n o número total de indivíduos no *ranking* (Kendall, 1990). Por exemplo, a Tabela 3.2 dispõe 10 indivíduos e dá seus *ranks* com relação a uma medida hipotética A . Nessa distribuição, cada indivíduo recebeu como *rank* um valor numérico que é providencialmente igual à sua classificação no *ranking*, de modo que, para cada indivíduo cujo *rank* é i , existem outros $i - 1$ indivíduos com melhor *rank* e outros $n - i$ indivíduos com *rank* inferior ou igual. No entanto, não se sabe o quão melhor ou pior são esses indivíduos, pois a aplicação do *ranking* descarta a magnitude das medidas, considerando apenas o relacionamento entre os indivíduos.

Tabela 3.2: *Ranking* envolvendo 10 indivíduos.

Indivíduos	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
Medida (A)	0.2	0.9	0.4	0.2	0.6	0.2	0.9	0.5	1.0	0.7
<i>Rank</i> (A)	8	2	7	8	5	8	2	6	1	4

Para melhor ilustrar, o *ranking* apresentado na Tabela 3.2 foi reorganizado de modo que os indivíduos estejam ordenados segundo seus *ranks*. Como resultado, obtém-se a Tabela 3.3 apresentada a seguir.

Tabela 3.3: *Ranking* posicional.

Indivíduos	I_9	I_7	I_2	I_{10}	I_5	I_8	I_3	I_1	I_6	I_4
Medida (A)	1.0	0.9	0.9	0.7	0.6	0.5	0.4	0.2	0.2	0.2
<i>Rank</i> (A)	1	2	2	4	5	6	7	8	8	8

¹Neste trabalho, adota-se os termos originais em inglês por serem os mais conhecidos da comunidade acadêmica.

Observe que, no caso de empate, todos os indivíduos empatados recebem como *rank* o menor valor de posição que aqueles indivíduos receberiam caso não houvesse nenhuma estratégia de modificação dos *ranks* em caso de empate e o indivíduo “seguinte” recebe como *rank* o valor de sua posição. Assim, os indivíduos I_7 e I_2 recebem o *rank* 2 e o indivíduo I_{10} recebe o *rank* 4. Para fins de comunicação, tal *ranking* é chamado neste texto de *ranking* posicional.

O *ranking* posicional é a forma usual de se representar *rankings* em veículos informais. Entretanto, para os propósitos de aprendizado de máquina é frequentemente desejável que a somatória dos *ranks* seja sempre uma constante dependente do número de indivíduos representados no *ranking* – a saber, $\frac{n^2+n}{2}$. Por isso, é comum que os *ranks* dos indivíduos empatados sejam trocados pela média dos valores dos *ranks* que os indivíduos receberiam caso não houvesse nenhuma estratégia de modificação dos *ranks* em caso de empate. Por exemplo, os indivíduos I_1 , I_6 e I_4 , sem nenhuma estratégia de empate, receberiam os *ranks* 8, 9 e 10, não necessariamente respectivamente. Nessa estratégia, os três indivíduos recebem o *rank* 9. Para fins de comunicação, este *ranking* é chamado neste texto de *ranking* normalizado. Na Tabela 3.4 é apresentado o *ranking* normalizado equivalente aos *rankings* apresentados nas tabelas anteriores.

Tabela 3.4: *Ranking* normalizado. Atente para a presença de *ranks* não-inteiros.

Indivíduos	I_9	I_7	I_2	I_{10}	I_5	I_8	I_3	I_1	I_6	I_4
Medida (A)	1.0	0.9	0.9	0.7	0.6	0.5	0.4	0.2	0.2	0.2
Rank (A)	1	2.5	2.5	4	5	6	7	9	9	9

Formalmente, a definição de um *ranking* normalizado baseado em um critério mensurável é a seguinte. Seja $\mathbf{I}$ um conjunto de n indivíduos e seja $\mathbf{M}$ um conjunto de valores relacionados a $\mathbf{I}$. Considere que o valor $m_i \in \mathbf{M}$ é um indicativo de qualidade do indivíduo i , de forma que, se $m_a > m_b$, então o indivíduo a é considerado “melhor” que o indivíduo b sob algum aspecto. Assim, o vetor $R = (r_1, \dots, r_n)$ é um *ranking* normalizado se estabelece uma ordem parcial sobre os indivíduos tal que, dado qualquer par de indivíduos $I_a, I_b \in \mathbf{I}$, sempre que $m_a > m_b$, então $r_a < r_b$ e é possível verificar que $\sum_{i=1}^n r_i = \frac{n^2+n}{2}$.

3.5 Composição de Rankings

Quando se possui um conjunto de indivíduos classificados em diferentes *rankings* segundo diferentes critérios, pode-se construir um *ranking* composto atribuindo-se a cada indivíduo um *rank* que reflete seu desempenho com relação aos demais quando se considera todos os critérios simultaneamente. Por exemplo, dado um conjunto de jogadores de futebol, pode-se classificar os jogadores por velocidade, precisão de chute, habilidade de drible etc. Um *ranking* composto seria um *ranking* tal que in-

divíduos melhores classificados nesse *ranking* poderiam ser considerados melhores jogadores em um sentido geral.

A utilização de composição de *rankings* é interessante para a simplificação de problemas de otimização multiobjetivo porque é uma forma de compor as funções objetivo que não envolve a composição de funções cujos valores são representados por grandezas potencialmente incompatíveis, em contraste com o método baseado na média ponderada das funções apresentado na seção anterior, não exigindo assim a definição de pesos, o que, dependendo do domínio, poderia ser altamentbte complexo ou mesmo impossível.

O procedimento geral de composição de *rankings* é o seguinte. Dado um problema de otimização multiobjetivo com m objetivos distintos, define-se inicialmente m *rankings* normalizados distintos $R_1, R_2, \dots, R_m$, cada um relacionado a uma função objetivo f_i . Em seguida, para cada indivíduo $I_i, i \in [1, n]$, cujos *ranks* nos diferentes *rankings* são dados por $r_{1i}, r_{2i}, \dots, r_{mi}$, é definido um valor de *rank* composto r_i que reflete a eficiência da solução com relação às demais considerando-se todos os objetivos simultaneamente.

Um exemplo desse procedimento é dado na Tabela 3.5. Inicialmente, todas as soluções são *rankeadas* segundo dois objetivos, quantificados pelas funções $(\max) f_1(\mathbf{x})$ e $(\max) f_2(\mathbf{x})$. Com base nisso são construídos os *rankings* normalizados R_1 e R_2 . Então, para cada solução é calculado um *rank* composto equivalente ao valor médio dos *ranks* obtidos por essa solução nos *rankings* iniciais. O resultado final é o *ranking* R_c , não necessariamente normalizado, que é convertido no *ranking* final R_f , este sim normalizado. Por exemplo, a solução S_2 atende aos objetivos com $f_1(S_2) = 9.6$ e $f_2(S_2) = 9.8$, obtendo os *ranks* 3.5 e 1 em R_1 e R_2 , respectivamente. Seu *rank* composto, portanto, será 2.25.

Tabela 3.5: Exemplo de composição de *rankings* pelo método da média aritmética.

Soluções	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
Função objetivo (f_1)	10.3	9.6	9.8	9.6	8.0	8.7	8.7	7.1	6.0	6.1
<i>Ranking</i> (R_1)	1	3.5	2	3.5	7	5.5	5.5	8	10	9
Função objetivo (f_2)	9.0	9.8	6.6	7.3	5.6	5.6	5.1	4.7	4.7	4.0
<i>Ranking</i> (R_2)	2	1	4	3	5.5	5.5	7	8.5	8.5	10
<i>Ranking</i> composto (R_c)	1.5	2.25	3	3.25	6.25	5.5	6.25	8.25	9.25	9.5
<i>Ranking</i> final (R_f)	1	2	3	4	6.5	5	6.5	8	9	10

É importante ressaltar, entretanto, que a composição de *rankings* não substitui com perfeição a comparação das soluções por dominância. Por exemplo, na Tabela 3.5, observa-se que não existe relação de dominância entre S_1 e S_2 . Apesar disso, como a composição de *rankings* é afetada pela presença de outras soluções no *ranking*, solução S_1 é considerada superior à solução S_2 , quando, na verdade, as soluções S_1 e S_2 compõem a fronteira de Pareto do conjunto de soluções apresentado. A aplicação de um método de otimização simples-objetivo que considera o *rank* composto como única

função-objetivo e procura encontrar uma única solução ótima certamente resultará no descarte de muitas soluções que seriam consideradas ótimas no contexto original. Uma forma de reduzir o impacto deste problema consiste em utilizar otimização simples-objetivo multimodal, buscando encontrar não apenas uma solução ótima ou todas as soluções ótimas de igual *rank*, porém um misto de soluções ótimas e soluções sub-ótimas.

Existe uma ampla variedade para composição de *rankings*. Na subseções a seguir, alguns métodos de composição de *rankings* utilizados neste trabalho são apresentados. Observe que a questão da composição de *rankings* é tratada neste capítulo apenas como uma forma de conversão do problema multiobjetivo em problema simples-objetivo. Uma discussão sobre a utilização desses métodos em um contexto de otimização multiobjetivo e de outros métodos de solução de problemas de otimização é realizada na Seção 3.6.

3.5.1 Médias aritmética e harmônica ponderadas

A composição de *rankings* por média é um método simples que calcula os *rank*s compostos R_{ci} através de uma função $Cr(R, W)$, sendo $R_r = (r_{1i}, r_{2i}, \dots, r_{mi})$ um vetor de *rank*s associados ao indivíduo I_i e $W = (w_1, \dots, w_m)$ um vetor de pesos associados aos objetivos ($W \in \mathbb{R}^m$).

O peso pode ser utilizado para privilegiar ou penalizar um ou mais objetivos na consideração do *ranking* composto. A distinção entre o *ranking* composto pela média harmônica ou pela média ponderada consiste na escolha da função $Cr(R, W)$. Como exemplo, duas funções são apresentadas a seguir: uma função baseada na média aritmética ponderada (Equação 3.3) e uma função baseada na média harmônica ponderada (Equação 3.4).

$$Cr(R, W) = \frac{\sum (w_i \times r_i)}{\sum w_i} \quad (3.3)$$

$$Cr(R, W) = \frac{\sum w_i}{\sum \frac{w_i}{r_i}} \quad (3.4)$$

A utilização de pesos é opcional e pode ser utilizada quando o usuário possui conhecimento adicional a respeito do domínio. Por exemplo, caso o usuário queira maximizar duas funções $f_1(\mathbf{x})$ e $f_2(\mathbf{x})$, porém privilegiando sutilmente soluções que maximizem a função $f_1(\mathbf{x})$, ele pode escolher como pesos o vetor $W = (5, 4)$. Note que a definição de pesos para a composição pela média não é uma condição necessária para encontrar soluções ótimas, mas um parâmetro opcional, não oferecendo a mesma dificuldade que existe quando se compõe uma função $F(f_1, f_2, \dots, f_m)$ sem a utilização de *rankings*.

3.5.2 Mediana

Outra forma de selecionar um valor de *rank* composto para uma solução é calcular a mediana de seus *ranks*. A mediana é uma estatística obtida quando se ordena um conjunto de dados e se escolhe o valor do dado que ocupa o índice central. O processo subjacente ao método da composição pela mediana é definir um *rank* composto tal que, no caso médio, a probabilidade de que o *rank* composto seja inferior ou superior aos *ranks* originais é mínima, independente dos valores de *ranks* obtidos pela solução de acordo com os objetivos.

Um exemplo de aplicação da composição pela mediana é dado na Tabela 3.6. Observe que a tabela não mostra um *ranking*, mas sim os *ranks* atribuídos a uma solução I_i qualquer com relação a cinco funções objetivos distintas, $f_1(\mathbf{x})$, $f_2(\mathbf{x})$, $f_3(\mathbf{x})$, $f_4(\mathbf{x})$ e $f_5(\mathbf{x})$. Os *ranks* são apresentados em ordem crescente por conveniência. A mediana, 7, é escolhida como o *rank* composto.

Tabela 3.6: *Ranks* atribuídos a uma solução I_i qualquer.

Medidas	$f_3(\mathbf{x})$	$f_1(\mathbf{x})$	$f_4(\mathbf{x})$	$f_2(\mathbf{x})$	$f_5(\mathbf{x})$
<i>Ranks</i>	4	6	7	8	8

Observe que, de acordo com a própria definição de mediana, quando o número de objetivos for par, existem dois possíveis valores de *rank* composto. Neste caso, a ação ideal consiste em selecionar uma das duas medianas aleatoriamente ou tomar a média de ambas como valor do *rank* composto para evitar a inserção de viés na composição do *ranking*.

3.5.3 O *Ranking* de Condorcet

O método da composição pelo *ranking* de Condorcet estabelece uma ordem parcial entre um conjunto de soluções e define um *ranking* com base nessa ordem. Essa ordem parcial é calculada em dois passos, sendo o primeiro realizado após uma comparação de todos os pares que podem ser obtidos do conjunto de soluções e o segundo após uma comparação global entre as soluções. Existem diferentes implementações do *ranking* de Condorcet, as quais na maioria dos casos diferem na forma como o segundo passo é conduzido. Aqui será apresentada a estratégia utilizada neste trabalho.

O primeiro passo de aplicação do *ranking* de Condorcet como adotado neste trabalho consiste em comparar as soluções par-a-par, verificando quais soluções são superiores no maior número de objetivos. Dado um par de soluções (S_a, S_b) , para cada função objetivo f_j é atribuído um ponto à solução S_a se esta é melhor que S_b com relação a f_j . Observe que isso não significa necessariamente $f_j(S_a) > f_j(S_b)$, pois existe a possibilidade de que a função se relacione a um objetivo de minimização. Ao se considerar todos os objetivos, se uma das soluções contabilizar uma pontuação maior que a ou-

tra, ela é declarada vencedora. Caso contrário, considera-se que houve um empate entre ambas. Não há necessidade de contabilizar derrotas.

Para exemplificar, na Tabela 3.7 são apresentados dez indivíduos e seus valores com relação a três objetivos, representados pelas funções $(max) f_1(\mathbf{x})$, $(min) f_2(\mathbf{x})$ e $(min) f_3(\mathbf{x})$. Analisando-se, por exemplo, o par (S_1, S_3) , verifica-se que a solução S_1 é vencedora, pois ela é superior a S_3 em todos os objetivos. Por outro lado, do par (S_5, S_6) declara-se um empate, pois ambas são igualmente boas com relação ao primeiro objetivo e cada uma é melhor que a outra com relação a um dos outros dois objetivos.

Tabela 3.7: Soluções comparadas pelo *ranking* de Condorcet.

Soluções	S_3	S_5	S_6	S_2	S_1	S_9	S_8	S_4	S_7	S_{10}
$(max) f_1(S_i)$	5.4	5.0	5.0	5.0	4.2	2.9	4.0	2.9	3.3	2.0
$(min) f_2(S_i)$	0.2	0.2	0.9	0.6	0.5	1.0	3.8	1.5	2.0	1.5
$(min) f_3(S_i)$	1.8	1.3	1.0	1.8	2.2	4.7	5.5	3.0	4.8	6.0

Comparando-se todos os pares factíveis (S_a, S_b) de soluções, com a garantia de $a < b$ para evitar que os mesmos indivíduos sejam comparados repetidamente, obtêm-se o número total de vitórias e o número total de derrotas de cada solução. Prosseguindo com os indivíduos da Tabela 3.7, na Tabela 3.8 a seguir são apresentados esses valores totais.

Tabela 3.8: Vitórias e empates contabilizados para as soluções da Tabela 3.7

Soluções	S_3	S_5	S_6	S_2	S_1	S_9	S_8	S_4	S_7	S_{10}
Vitórias	9	7	7	6	5	2	2	1	1	0
Empates	0	1	0	1	0	2	2	2	2	0

O passo final consiste em construir um *ranking* com base nos valores totais de vitórias e empates, utilizando o número de vitórias para estabelecer uma ordem parcial entre as soluções e o número de empates como critério de distinção entre as soluções com mesmo número total de vitórias. As soluções com maior número de vitórias e, como segundo critério, de empates, recebem os *ranks* iniciais. As soluções com mesmo número de vitórias e empates recebem *ranks* empatados – e normalizados.

Na Tabela 3.9 a seguir são apresentados os *ranks* para as soluções consideradas nos exemplos anteriores. Observe que as soluções S_5 e S_6 , embora apresentem igual número de vitórias, não são consideradas empatadas nesta fase do *rankeamento* porque o número de empates constatados na fase anterior para as duas soluções é diferente. Já as soluções S_9 e S_8 e as soluções S_4 e S_7 empatam, pois obtiveram na fase anterior igual número de vitórias e empates.

Tabela 3.9: Composição final do *ranking* de Condorcet.

Soluções	S_3	S_5	S_6	S_2	S_1	S_9	S_8	S_4	S_7	S_{10}
Vitórias	9	7	7	6	5	2	2	1	1	0
Empates	0	1	0	1	0	2	2	2	2	0
<i>Ranking</i>	1	2	3	4	5	6.5	6.5	8.5	8.5	10

3.5.4 O *Ranking* Recíproco

No *Ranking* recíproco, o *rank* composto é obtido através da soma dos inversos dos *ranks* originais, de acordo com a Equação 3.5. Como anteriormente, tem-se que $R_r = (r_{1i}, r_{2i}, \dots, r_{mi})$ é um vetor de *ranks* atribuídos à solução S_i e $W = (w_1, \dots, w_m)$ é um vetor de pesos associados às medidas ($W \in \mathbb{R}^m$).

$$Rc(R, W) = \sum_{i=1}^M \frac{w_i}{r_i} \quad (3.5)$$

Observe que, por se tratar de uma soma dos inversos, o *ranking* produzido por este método também deve ser ordenado de forma inversa. Isto é, quanto maior o valor do *rank* composto, melhor a solução com relação a todos os objetivos. Por exemplo, na Tabela 3.10 são mostradas dez soluções e seus valores segundo três funções objetivos, $(\max) f_1(\mathbf{x})$, $(\min) f_2(\mathbf{x})$ e $(\min) f_3(\mathbf{x})$. Para cada função f_j é definido um *ranking* R_j . O *ranking* composto é definido pelo vetor R_c . Um *ranking* final normalizado R_f é construído com base nos *ranks* do *ranking* composto.

Tabela 3.10: Exemplo completo do processo de composição por *ranking* recíproco

Soluções	S_3	S_5	S_6	S_2	S_1	S_9	S_4	S_8	S_7	S_{10}
Função Objetivo (f_1)	5.4	5.0	5.0	5.0	4.2	2.9	2.9	4.0	3.3	2.0
<i>Ranking</i> (R_1)	1	3	3	3	5	8.5	8.5	6	7	10
Função Objetivo (f_2)	0.2	0.2	0.9	0.6	0.5	1.0	1.5	3.8	2.0	1.5
<i>Ranking</i> (R_2)	1.5	1.5	5	4	3	6	7.5	9	8	7.5
Função Objetivo (f_3)	1.8	1.3	1.0	1.8	2.2	4.7	3.0	5.5	4.8	6.0
<i>Ranking</i> (R_3)	3.5	2	1	3.5	5	7	6	9	8	10
<i>Ranking</i> composto (R_c)	1.95	1.5	1.5	0.87	0.73	0.43	0.42	0.39	0.39	0.33
<i>Ranking</i> final (R_f)	1	2.5	2.5	4	5	6	7	8.5	8.5	9

3.6 Métodos de Resolução

Um grande número de métodos de resolução existe à disposição para lidar com problemas de otimização simples-objetivo e multiobjetivo. Em muitos casos, os métodos para otimização simples-objetivo podem ser utilizados para solucionar problemas de otimização multiobjetivo, desde que mecanismos para conversão do problema de otimização sejam empregados, como os métodos de composição de *rankings* apre-

sentados neste capítulo. Entretanto, conforme discutido na Seção 3.5, nem sempre uma composição dos objetivos permite encontrar uma fronteira de Pareto razoável, *i.e.*, uma cujas soluções pertençam à fronteira Pareto-ótimo ou sejam “próximas” das soluções presentes na fronteira Pareto-ótima e que forneçam uma variedade razoável de soluções não-dominadas para o usuário considerar.

Em geral, métodos de otimização, tanto no caso simples-objetivo quanto no caso multi-objetivo, tentam solucionar o problema de otimização encontrando um conjunto de soluções não necessariamente ótimas e modificando essas soluções de modo a gerar soluções potencialmente melhores. Diferentes vieses adotados pelos métodos para encontrar soluções iniciais, modificar essas soluções, compará-las de acordo com sua eficiência e determinar o momento no qual as soluções não devem mais ser modificadas – critério de parada – caracterizam e distinguem os métodos uns dos outros.

Uma forma muito simples de solucionar problemas de otimização simples-objetivo e multi-objetivo é a enumeração exaustiva. Trata-se de uma aplicação direta de algoritmo de força bruta e consiste em enumerar todas as possíveis combinações de valores para as variáveis de decisão. Um ponto positivo deste método é que, se todas as variáveis de decisão forem discretas e pertencerem a um domínio finito, é garantido que a solução ótima ou todas as soluções da fronteira Pareto-ótima serão encontradas. No caso em que houver variáveis de decisão contínuas, pode-se discretizar o domínio dessas variáveis, ao custo da garantia de se encontrar soluções ótimas. É fácil perceber que a enumeração exaustiva exige a implementação de um algoritmo com complexidade exponencial, equivalente ao problema de se enumerar todos os subconjuntos de um conjunto qualquer. Por essa razão, enumeração exaustiva raramente é utilizada.

O processo de otimização pode ser facilmente comparado a um método de busca em um espaço de variáveis. Na Figura 3.4 é ilustrado um espaço de decisão simplificado. O eixo das abcissas representa pontos do espaço de decisão. Cada ponto nesse eixo corresponde a um vetor $\mathbf{x}$ de dimensionalidade N , correspondente a N variáveis de decisão, que foi de alguma forma simplificado para uma reta por um processo de redução de dimensionalidade qualquer. O eixo das coordenadas representa uma função objetivo $f(\mathbf{x})$. Considere que o objetivo da otimização é maximizar $f(\mathbf{x})$.

Encontrar uma solução qualquer é encontrar um vetor $\mathbf{x}$ que pode ser mapeado em algum ponto no eixo das abcissas. Na Figura 3.4 (a) o círculo marca a posição que uma solução $\mathbf{x}_i$ qualquer ocupa na curva $f(\mathbf{x})$, com a “altura” representando o valor de $f(\mathbf{x}_i)$. Quando essa solução é de alguma forma modificada, encontra-se um novo ponto $\mathbf{x}_j$ e um novo ponto na curva. Essa transição é representada pela sequência de (a) para (b) na Figura 3.4.

A enumeração exaustiva não utiliza nenhuma informação para guiar o processo de busca. Portanto, ela é equivalente a uma busca cega. Outros métodos de busca, como o *hill climbing* e o *branch and bound* (Russell e Norvig, 2003) podem ser utilizados.

Figura 3.4: Exemplo de busca no espaço de decisão.

Esses e outros métodos, ao utilizar informação da função objetivo, são capazes de descartar ou privilegiar regiões do espaço de busca se for improvável ou provável que essas regiões contêm a solução ótima.

Por exemplo, o algoritmo *hill climbing*, cujo nome pode ser traduzido para o português como “subida da encosta”, explora o espaço de busca analisando a vizinhança de uma solução na curva $f(\mathbf{x})$, como pode ser observado na Figura 3.5 (a), no qual o círculo representa uma solução considerada pelo algoritmo de busca. As setas indicam soluções da vizinhança que o algoritmo irá considerar para realizar a transição. Se alguma das soluções estiver situada em um ponto da curva tal que se percebe uma melhor otimização da função objetivo, o algoritmo passa a considerar essa solução e descarta a solução original. Se a solução na vizinhança da solução original não estiver em uma região da curva na qual a função objetivo é melhor otimizada, então essa solução não é considerada. Essa transição pode ser observado na sequência de (a) para (b) na Figura 3.5. Esse processo de escolha por pontos mais “altos” da curva lembra alguém “subindo uma encosta”, daí o nome do algoritmo.

Figura 3.5: Exemplo de transição do *hill climbing*.

A busca termina quando não houver nenhum ponto na vizinhança de nenhuma solução considerada pelo algoritmo tal que pode-se verificar pela curva que a função

objetivo é melhor maximizada. A isso dá-se o nome de *convergência*. Extrapolando-se o exemplo da Figura 3.5, é fácil perceber que, se o processo for repetido número suficiente de vezes, o algoritmo irá convergir para o ponto indicado pela estrela – o máximo global.

O procedimento utilizado neste trabalho para otimização multiobjetivo segue princípio semelhante. Em um determinado momento o algoritmo considera um conjunto de soluções $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ no espaço de objetivos. Essas soluções são então de alguma forma modificadas e as posições ocupadas por essas novas soluções no espaço de objetivos são verificadas. Com base na dominância entre as soluções, o algoritmo pode encontrar a fronteira de Pareto e separar as soluções em dois conjuntos: um contendo soluções sub-ótimas e outro contendo soluções potencialmente ótimas. Utilizando essa informação o algoritmo pode descartar algumas soluções, preferencialmente soluções sub-ótimas, estabelecer um novo conjunto de soluções que potencialmente é melhor, em termos de eficiência de Pareto, que o conjunto de soluções inicial. A iteração desse processo leva o algoritmo a considerar conjuntos cada vez mais eficientes, até o momento em que um conjunto “bom o bastante” – preferencialmente um próximo da fronteira Pareto-ótimo e com alta variedade de opções – seja encontrado. Nesse ponto é declarada convergência e o conjunto de soluções final pode ser apresentado ao usuário.

Um exemplo da transição envolvida nesse processo é mostrado na Figura 3.6. A figura ilustra o espaço de objetivos de um problema de otimização (max, max) qualquer. Tanto em 3.6 (a) quanto em (b) os quadrados e os círculos indicam os pontos ocupados por diferentes soluções no espaço de objetivos. A linha contínua no “canto” do espaço de objetivos representa a fronteira Pareto-ótima.

Figura 3.6: Exemplo de transição de um algoritmo de otimização multiobjetivo.

Em Figura 3.6 (a), os círculos indicam o conjunto de soluções original, composto por sete soluções distintas, cuja fronteira de Pareto é demarcada pela linha tracejada. Para realizar a transição de (a) para (b), o algoritmo de otimização inicialmente seleciona algumas soluções não-dominadas do conjunto original e as modifica, produzindo quatro novas soluções representadas pelos quadrados. Desse novo conjunto com-

posto por 11 soluções, aquelas que são dominadas são representadas pelos círculos e quadrados sem borda. O algoritmo então descarta quatro soluções, preferencialmente dominadas, as quais são marcadas em 3.6 (b) com $\mathbf{x}$. O novo conjunto composto por sete soluções é considerado em uma nova iteração do mesmo processo. E assim sucessivamente até que um conjunto de soluções “bom o bastante” seja encontrado.

Métodos de busca como *hill climbing* e aqueles baseado no conceito de eliminação iterativa de regras dominadas são mais eficientes que a enumeração exaustiva porque descartam significativamente trechos do espaço de decisão sem avaliar todas as soluções contidas nesses trechos. Isso reduz substancialmente a complexidade computacional da busca. Entretanto, isso também insere o risco de que a busca seja interrompida prematuramente se a solução inicial ou o conjunto de soluções iniciais estiver próximo de uma “encosta” que não leva ao máximo global, mas a um máximo local da função. Isso é chamado *convergência prematura* e pode ser particularmente problemático em domínios mais complexos, que podem conter máximos e mínimos locais, bem como platôs e descontinuidades – *e.g.* Figura 3.7. Equivalentemente, no caso do problema de otimização multiobjetivo, se o espaço de busca – omitido nos exemplos apresentados – for complexo, o algoritmo pode ter dificuldade em encontrar soluções que dominem as soluções consideradas em uma iteração qualquer, o que implica que o algoritmo terá dificuldade em encontrar um conjunto de soluções próximas da fronteira Pareto-ótima (Deb, 2001).

Figura 3.7: Exemplo de uma função objetivo com máximos e mínimos locais.

Existem diversas técnicas de otimização simples-objetivo e multiobjetivo, com diferentes abordagens para evitar a convergência prematura em máximos locais. Um conjunto de técnicas e métodos particularmente interessante para a resolução de problemas de otimização em espaços de busca complexos são as técnicas de computação evolutiva e algoritmos genéticos. Essas técnicas promovem uma busca global do espaço de decisão, podendo ser utilizada tanto para otimização simples-objetivo quanto otimização multiobjetivo. Computação evolutiva e algoritmos genéticos, bem como a aplicação desses métodos para a busca por regras com propriedades desejadas, são discutidos nos próximos capítulos.

3.7 Considerações Finais

Neste capítulo foram introduzidos conceitos sobre otimização simples-objetivo e multiobjetivo, bem como a utilização de *rankings* para a conversão de problemas de otimização multiobjetivo em simples-objetivo. O conceito de otimização multiobjetivo é de grande importância para este trabalho, pois estabelece um método de resolução para o problema de se construir regras de conhecimento com propriedades de interesse do usuário quando a descoberta de regras que satisfaçam essas propriedades é interpretada como um problema de otimização no qual uma métrica referente a cada propriedade é uma função objetivo.

Como mencionado anteriormente, o problema de otimização multiobjetivo envolvido neste trabalho é solucionado com a aplicação de um algoritmo de computação evolutiva multiobjetivo. Assim, no próximo capítulo desta Dissertação são introduzidos conceitos fundamentais de computação evolutiva e algoritmos genéticos. Métodos de resolução específicos a problemas de otimização multiobjetivo, tal como NSGA-II (Deb, 2001), são abordados posteriormente no Capítulo 5. Ainda, no Capítulo 5 é realizada uma avaliação de algoritmos genéticos e rankings, bem como de outros métodos de otimização multiobjetivo, no domínio de indução de regras com propriedades desejadas.

Computação Evolutiva e Algoritmos Evolutivos

4.1 Considerações Iniciais

Computação evolutiva refere-se a um conjunto de técnicas de otimização baseadas nos conceitos de evolução, seleção natural, macroevolução e adaptação ao ambiente ([Mitchell, 1998](#)) para realizar uma busca global em um espaço de variáveis por meio de um processo de evolução artificial. Simplificadamente, pode-se afirmar que algoritmos baseados em computação evolutiva tipicamente consistem na modificação iterativa e aleatória de um conjunto de soluções, favorecendo as modificações que produzem soluções melhores e desfavorecendo as demais.

A computação evolutiva é um campo vasto que tem sido estudado desde meados do século XX para solucionar problemas de otimização. Dentre as muitas técnicas que podem ser consideradas técnicas de computação evolutiva, encontram-se os algoritmos evolutivos, os quais se caracterizam por representar soluções como indivíduos competindo entre si pela oportunidade de transmitir suas características. Neste trabalho, algoritmos evolutivos utilizados para construção de regras de conhecimento permitirem estabelecer um processo de busca robusto e facilmente adaptável ao domínio de aplicação.

A estrutura deste capítulo é como se segue: na Seção [4.2](#) são expostos alguns conceitos fundamentais de computação evolutiva. Na Seção [4.3](#) os conceitos formais de algoritmos evolutivos são esclarecidos. Nas Seções [4.4](#) a [4.9](#), incluindo respectivas subseções, são apresentados os elementos necessários para a implementação de um algoritmo evolutivo, bem como alguns conceitos adicionais pertinentes. Finalmente, algumas considerações finais são traçadas na Seção [4.10](#).

4.2 Computação Evolutiva

Computação evolutiva refere-se a um conjunto de técnicas de IA que utilizam conceitos de evolução para solucionar problemas de otimização. Os primeiros trabalhos relacionados a computação evolutiva datam da década de 1950, porém apenas a partir da década de 1980 a área passou a gozar de popularidade entre pesquisadores do meio acadêmico (Bäck et al., 1997). Dentre as principais vertentes de computação evolutiva em uso na atualidade, pode-se citar as técnicas baseadas em inteligência de colônia e os algoritmos evolutivos, dentre outras.

Os algoritmos baseados em inteligência de colônia utilizam um conjunto de agentes simples que operam de forma descentralizada, interagindo entre si e com o ambiente. Apesar da simplicidade dos agentes, a complexidade da interação pode levar os agentes a evocar uma espécie de “inteligência coletiva”. Essa interação é interpretada pelo algoritmo como uma possível solução para o problema de otimização, a qual é melhorada a cada iteração do algoritmo até que algum critério de parada seja satisfeito (Dorigo, 1992).

A colônia artificial de formigas (Dorigo et al., 2006; Blum, 2005) é provavelmente a técnica mais conhecida de inteligência de colônia. A colônia artificial de formigas mapeia o problema de otimização para um ambiente virtual e uma simulação de formigas explorando esse ambiente é promovida. Nos termos da simulação, as formigas interagem entre si espalhando feromônio pelo espaço que percorrem e interagem com o ambiente de diversas formas. Cada formiga toma decisões que a levam a explorar um espaço desse ambiente com base na quantidade de feromônio deixado por outras formigas e com base no reforço dado pelo ambiente às formigas, acrescidos de um fator de aleatoriedade. Por exemplo, a proximidade de um alimento aumenta a probabilidade de uma formiga visitar uma certa região do espaço, assim como uma maior concentração de feromônio em um espaço aumenta a probabilidade de que a formiga decida visitar esse espaço. Com o tempo, os espaços que as formigas “preferem” percorrer definem caminhos que codificam uma solução possivelmente ótima do problema.

Algoritmos evolutivos, por sua vez, utilizam indivíduos para representar potenciais soluções do problema, isto é, soluções que podem ser factíveis ou não e, sendo factíveis, podem ser ótimas ou não. Cada indivíduo representa uma única solução potencial. Um conjunto de soluções representadas por indivíduos em um dado momento é chamado população. O algoritmo evolutivo é iterativo, modifica a população a cada etapa do processo, privilegiando os indivíduos que representam melhores soluções, de forma que a população construída em uma iteração tende a conter soluções melhores que aquelas da iteração anterior. Esse processo de modificação prossegue até que a solução ótima ou fronteira Pareto-ótima seja encontrada ou outro critério de parada seja satisfeito.

As seções restantes deste capítulo focam exclusivamente em algoritmos evolutivos, ainda que algumas das noções apresentadas a seguir possam ter aplicações também em outras técnicas de computação evolutiva.

4.3 Algoritmos Evolutivos

Algoritmos evolutivos são um caso particular de computação evolutiva que têm como característica principal a representação das soluções como indivíduos que competem em um espaço virtual para propagar suas características. Algoritmos evolutivos são interessantes a este trabalho, em detrimento das demais técnicas de computação evolutiva, porque o conceito de indivíduo e a forma de avaliação dos indivíduos empregada pelo algoritmo evolutivo é conveniente para a construção de regras de conhecimento individuais baseada em critérios arbitrários.

Algoritmos evolutivos implementam um processo de seleção artificial que remete aos conceitos da teoria da evolução pela seleção natural. Assim como na natureza cada espécime é “testado” pelo ambiente e os espécimes mais aptos tendem a sobreviver, no algoritmo evolutivo os indivíduos são “testados” computacionalmente e aqueles que representam soluções melhores tendem a permanecer na população, ao passo que os indivíduos restantes tendem a ser descartados.

Esse processo de seleção artificial necessita de um formalismo de representação e funcionamento que simula computacionalmente, com certa liberdade, o que ocorre na natureza (Michalewicz, 1997). A cada iteração $t = 0, 1, 2, \dots$ o algoritmo considera uma população de indivíduos $P(t)$ que *representa* um conjunto de soluções potenciais. Para simular a competição entre os indivíduos, uma *função de avaliação* testa a eficácia da solução representada por cada indivíduo e atribui ao indivíduo um valor de *aptidão*. A informação de aptidão é utilizada por um mecanismo de seleção estocástico, o *operador de seleção*, que seleciona de forma aleatória, porém informada, um conjunto de indivíduos privilegiando os melhores da população, compondo uma *população intermediária* $P'(t)$. A população intermediária é então aleatoriamente modificada e uma nova população $P(t + 1)$ é construída. Esse ciclo de avaliação, seleção e modificação continua até que um indivíduo — ou um conjunto de indivíduos — que representa a solução ótima seja encontrado ou algum outro critério de parada seja satisfeito (Mitchell, 1998). Um algoritmo evolutivo típico é representado pelo Algoritmo 4.1.

Nas Seções 4.4 a 4.9 a seguir, os mecanismos que compõem um algoritmo evolutivo e os conceitos que os norteiam são explicados.

Algoritmo 4.1 Algoritmo evolutivo típico

```
1:  $t \leftarrow 0$ 
2: definir população inicial  $P(0)$ 
3: avaliar população inicial  $P(0)$ 
4: enquanto critério de parada não for satisfeito faça
5:   selecionar  $P'(t)$ 
6:   modificar  $P'(t)$ 
7:    $P(t+1) \leftarrow P'(t)$ 
8:    $t \leftarrow t+1$ 
9:   avaliar  $P(t)$ 
10: fim-enquanto
```

4.4 Indivíduos

O indivíduo é a base fundamental do algoritmo evolutivo. Cada indivíduo representa uma solução potencial para o problema de otimização e, desta forma, é necessária a definição de um esquema de codificação que permita que o algoritmo evolutivo possa manipular esse indivíduo e também ser capaz de converter uma solução em um indivíduo e vice-versa. É importante destacar que o esquema de codificação do indivíduo estabelece limites às manipulações que o algoritmo evolutivo poderá fazer, tanto em termos de poder de busca quanto em termos de eficiência computacional. Uma má escolha do esquema de codificação pode impedir que o algoritmo seja capaz de representar todas as soluções factíveis, incluindo soluções ótimas, bem como causar um *overhead* desnecessário na manipulação dos indivíduos, tornando o algoritmo evolutivo computacionalmente caro em excesso.

Diferentes linhas de algoritmos evolutivos utilizam diferentes abordagens de codificação de indivíduos e de operações associadas. Os algoritmos genéticos, introduzidos por (Holland, 1975) e popularizados por (Goldberg, 1989), são uma classe de algoritmos evolutivos que têm como principal característica a codificação de indivíduos como cadeias de *bits* ou variáveis de decisão. Contraste algoritmos genéticos com, por exemplo, programação genética e programação evolutiva, duas linhas de algoritmos evolutivos nas quais os indivíduos representam procedimentos ou algoritmos.

Uma hipótese acerca de algoritmos genéticos é a hipótese dos *blocos de construção*, a qual propõe que a razão pela qual os algoritmos genéticos são capazes de resolver problemas de otimização é que eles promovem a recombinação de “partes de informação” contidas nos cromossomos (Haupt e Haupt, 1998). Embora não seja uma hipótese comprovada ou corroborada por evidências “fortes” o bastante para ser considerada uma teoria, a hipótese dos blocos de construção encontra aceitação significativa entre pesquisadores da área e é bastante recorrente. De certa forma, a hipótese dos blocos de construção pode ser relacionada ao princípio de evolução centrada nos genes (Cronin, 1993), popularmente conhecido como princípio do “gene egoísta” (Dawkins, 1976). Segundo esse princípio, a seleção natural acontece essencialmente no nível genético e a evolução emerge da competição entre os genes, não

da competição entre os indivíduos, os quais seriam mera manifestação dos genes no espaço de fenótipos. Os genes que traduzem os melhores fenótipos para o ambiente no qual o indivíduo e o próprio gene se inserem tendem a assegurar ao indivíduo maiores chances de sobrevivência e, portanto, tendem a garantir maiores chances de perpetuação para si por meio da reprodução. De forma semelhante, os blocos de construção que, juntamente com outros blocos, codificam indivíduos que representam melhores soluções, tenderiam a ser favorecidos pelo algoritmo genético.

É comum que algoritmos genéticos codifiquem indivíduos como sequências de bits, cujos tamanhos e semânticas dependem das variáveis do problema de otimização. Alguns autores defendem que todo algoritmo genético utilize apenas sequências binárias na codificação dos indivíduos (Michalewicz, 1997). Entretanto, não são raras as situações nas quais codificações mais complexas são utilizadas para permitir que manipulações mais coerentes com as semânticas e os domínios das variáveis de decisão codificadas pelos indivíduos sejam feitas com menor custo computacional.

O algoritmo evolutivo utilizado neste trabalho é um algoritmo genético e codifica indivíduos capazes de representar soluções de acordo com a representação *Michigan*. Nessa representação, cada solução é uma regra de conhecimento. Contraste a representação *Michigan* com a representação *Pittsburgh*, na qual cada solução é um classificador.

Representação *Pittsburgh*: Na representação *Pittsburgh*, cada indivíduo da população representa um classificador baseado em regras, em contraste com a representação *Michigan*, na qual cada indivíduo da população representa uma única regra. Isso facilita a manipulação de indivíduos quando os blocos de construção considerados são regras de conhecimento, em vez de condições ou partes das condições dessas regras.

Tal paradigma é bastante vantajoso, conforme explicado por Freitas (2002), quando o propósito do algoritmo evolutivo é construir um classificador sobre todo o domínio dos dados. Considere que um classificador é bom se as regras que o compõem cobrem corretamente uma porção significativa do conjunto de dados, ao mesmo tempo que as regras apresentam pouca sobreposição. A sobreposição de regras é frequentemente indesejável, pois produz classificadores mais complexos contendo muita informação redundante. Um exemplo desse fenômeno é ilustrado na Figura 4.1. Na figura é ilustrado um espaço de exemplos contendo exemplos de duas classes e três regras referentes à classe negativa. Observe como existe sobreposição desnecessária das regras, que poderia ser evitada se as regras fossem mais concisas.

O uso da representação *Pittsburgh* permite que o algoritmo genético considere que os indivíduos são mais aptos quando as regras por eles representadas, além de cobrir corretamente um grande número de exemplos, possuem regras com menor sobreposição. Em contrapartida, a representação *Pittsburgh* não é

Figura 4.1: Exemplo de sobreposição de regras

tão eficiente quando existe interesse em otimizar regras individuais, pois os indivíduos representam regras de forma que tanto a avaliação da aptidão dos indivíduos quanto a manipulação dos indivíduos favorece a interação entre todas as regras que compõem o classificador.

Representação *Michigan*: Na representação *Michigan* os indivíduos de um algoritmo genético representam regras de conhecimento individuais e a população pode ser encarada como um conjunto de classificadores. Dependendo da aplicação, esse conjunto pode ser visto como um conjunto de cardinalidade unária – *i.e.*, todos os indivíduos juntos representam um único classificador – ou, no outro extremo, como um conjunto de cardinalidade igual ao tamanho da população – *i.e.*, cada indivíduo é uma regra completamente independente das demais.

Uma característica importante da representação *Michigan* é que ela permite a definição de um esquema de codificação que torna mais fácil a manipulação de indivíduos quando os blocos de construção são as condições das regras ou partes das condições das regras. Nesse sentido, a representação *Michigan* facilita a busca por regras de conhecimento que contenham as melhores condições segundo alguma medida de qualidade e é mais interessante para a construção de regras individuais.

Neste trabalho, a representação *Michigan* é utilizada porque não existe interesse na interação entre as regras de conhecimento construídas. Embora haja uma preocupação com a diversidade dos indivíduos, devido à necessidade de se apresentar ao usuário um conjunto amplo de regras de conhecimento boas e diversas, não existe uma preocupação de que todas as regras juntas apresentem alguma propriedade interessante, como cobrir bem todo o conjunto de exemplos.

É possível utilizar a representação *Michigan* tendo tais preocupações em mente. Entretanto, isso exige custos adicionais, pois o algoritmo genético tipicamente avalia os indivíduos isoladamente, sendo que a avaliação da população como um todo exige um custo adicional que poderia ser evitado com o uso a representação *Pittsburgh*.

Uma diferenciação adicional, porém não-obrigatória, entre as representações *Michigan* e *Pittsburgh* diz respeito ao tamanho do indivíduo. Algoritmos evolutivos podem codificar indivíduos com uma quantidade fixa ou variável de informação. Por exemplo, um indivíduo pode ser codificado como uma lista simplesmente encadeada ou como um vetor de tamanho fixo. Em ambos os casos, os elementos da lista ou do vetor podem ser elementos atômicos ou estruturas de dados mais complexas, podendo estas também possuir tamanho fixo ou variado. É comum, em algoritmos genéticos baseados na representação *Michigan*, que os indivíduos sejam codificados por vetores de tamanho fixo, ao passo que, em algoritmos genéticos baseados na representação *Pittsburgh*, é comum que os indivíduos sejam codificados como listas de tamanho variável, com cada elemento da lista representando uma regra do classificador.

Tal distinção é relevante porque codificar os indivíduos com quantidade fixa de informação facilita a implementação dos operadores de modificação – apresentados nas Seções 4.7 e 4.8. Por outro lado, uma codificação fixa limita o número de variáveis que podem ser associadas aos indivíduos. No caso da representação *Michigan*, em que os indivíduos representam regras de conhecimento, isso não é um problema, pois o número máximo de condições pode ser definido a partir do número de atributos do conjunto de dados, sendo apenas necessário um mecanismo para representar trechos do indivíduo que não codificam condições presentes na regra. No caso da representação *Pittsburgh*, entretanto, o número ideal de regras por classificador nem sempre pode ser definido, de modo que fixar um valor pode impedir soluções ótimas de serem representadas por indivíduos ou associar a cada indivíduo uma quantidade de informação desnecessariamente elevada, reduzindo o desempenho computacional do algoritmo.

Nas seções a seguir deste são abordados outros elementos fundamentais de um algoritmo genético. A maioria desses elementos se aplica também a algoritmos evolutivos em geral, assim, doravante não será feita distinção entre algoritmos genéticos e evolutivos. Observe que a codificação dos indivíduos não é necessária para a compreensão desses elementos, porém é importante ter em mente que cada indivíduo do algoritmo utilizado representa uma regra de conhecimento. Alguns detalhes de implementação, incluindo o esquema de codificação de indivíduos utilizado, serão abordados no Capítulo 5.

4.5 Função de Avaliação

Na natureza, de acordo com a teoria da seleção natural, o principal fator responsável pela evolução das espécies é o princípio da sobrevivência do mais apto. Todos os indivíduos estão constantemente competindo entre si, de modo que os indivíduos mais bem-adaptados ao ambiente tendem a encontrar maiores oportunidades de

procriação, ao passo que os indivíduos menos adaptados tendem a morrer antes de procriar.

No algoritmo evolutivo, dois componentes são responsáveis por promover um processo de seleção artificial que simula essa competição. Um desses componentes é o operador de seleção, o qual será discutido na próxima seção. O outro é a função de avaliação.

A *função de avaliação* é o primeiro passo de cada iteração de um algoritmo evolutivo típico. Ela analisa os indivíduos da população e atribui a cada um deles uma nota, chamada *aptidão* – ou *fitness*. É comum que a aptidão seja um valor no intervalo real $[0, 1]$, sendo que indivíduos com maior valor de aptidão são considerados mais *aptos* e, portanto, selecionados para as iterações seguintes com maior probabilidade. Entretanto, isso não é uma regra, podendo ser utilizado intervalo ou semântica distintos, contanto que os operadores do algoritmo seja modificados de acordo.

A função de avaliação deve ser formulada para atender às necessidades do domínio no qual o algoritmo evolutivo se aplica. É comum que o problema de otimização possa ser facilmente escrito como uma equação ou um conjunto de equações e, quando isso ocorre, as próprias equações podem ser utilizadas como funções de avaliação. Em alguns casos mais específicos, no entanto, a função de avaliação está implícita em uma simulação computacional, na avaliação de um experimento físico – *e.g.*, aprendizado de robôs sociais (Mitri et al., 2009) – ou na avaliação subjetiva de um usuário.

No caso de aprendizado de regras de conhecimento, a função de avaliação pode ser uma função envolvendo uma ou mais medidas de qualidade de regras, tais como aquelas apresentadas na Seção 2.5.1. A escolha da medida de qualidade depende exclusivamente dos objetivos do usuário em tratar o problema de otimização. Por exemplo, caso o usuário esteja buscando regras de conhecimento com altos graus de novidade e cobertura, pode-se utilizar uma função de avaliação que componha as medidas de qualidade de regra novidade e cobertura. Tal composição pode ser feita por cálculo de média ponderada – não se limitando a médias aritmética e harmônica – ou por composição de *rankings* – Seção 3.4.

É importante lembrar que problemas de otimização envolvendo mais de um objetivo podem se caracterizar como problemas de otimização multiobjetivo, bastando para isso que um dos objetivos seja conflitante com os demais. Nesse caso, utilizar como função de avaliação uma composição de funções que traduzam objetivos individuais é uma estratégia de redução do problema de otimização multiobjetivo para um problema de otimização simples-objetivo. Nem sempre tal redução é desejável, podendo ser preferível a utilização de estratégias dedicadas à otimização multiobjetivo. Nesse caso, em vez de uma única função de avaliação, o algoritmo evolutivo pode utilizar diversas funções de avaliação.

Para fins de compreensão, durante a leitura do restante deste capítulo, considere a situação na qual algum mecanismo é utilizado para atribuir a cada indivíduo um

único valor de aptidão, o qual pode ser obtido por meio de alguma composição, de um *ranking* produzido por alguma estratégia de otimização multiobjetiva ou por uma única função de avaliação aplicada a um problema de otimização simples-objetivo. Tal abstração facilita a compreensão dos operadores do algoritmo evolutivo sem perda de generalidade.

4.6 Operador de Seleção

O *operador de seleção* – também conhecido como *função de seleção* — é um mecanismo presente em algoritmos evolutivos que simula o ciclo de vida e morte encontrado na natureza. O operador de seleção é um dos elementos mais importantes do algoritmo evolutivo e, aliado à função de avaliação, permite selecionar com maior probabilidade os indivíduos mais aptos, propiciando a propagação de suas características, ao mesmo tempo que os indivíduos menos aptos são descartados.

No algoritmo evolutivo típico, o operador de seleção é o segundo passo de cada iteração t , sendo aplicado imediatamente após a função de avaliação. O papel do operador de seleção é amostrar um conjunto de indivíduos para compor uma população intermediária $P'(t)$. Essa população intermediária deve conter, idealmente, os indivíduos mais aptos da população, de modo que a construção da população $P(t + 1)$ a partir desses indivíduos permita que o algoritmo encontre soluções melhores.

Em essência, o operador de seleção age de forma aleatória, porém dependente da informação de aptidão dos indivíduos. Isto é, embora não seja possível determinar previamente quais indivíduos estarão presentes na população intermediária, é possível estimar essa informação de acordo com uma distribuição de probabilidades que depende dos valores de aptidão dos indivíduos. Apesar disso, o operador de seleção pode incluir algumas operações determinísticas, como aquelas baseadas no conceito de elitismo – Seção 4.9.

Assim como a função de avaliação, o operador de seleção deve ser designado de forma a explorar bem o domínio de aplicação. Diferentes métodos de seleção podem ser aplicados em um algoritmo evolutivo, sendo comum também que tais métodos contêm parâmetros que podem ser ajustados para modificar a aplicação da função de seleção.

Um importante parâmetro a ser considerado é a *pressão da seleção* (Haupt e Haupt, 1998), um fenômeno implícito a todo operador de seleção. Conforme explicado na Seção 4.5, quanto maior a aptidão de um indivíduo, maiores são suas chances de ser selecionado. A pressão de seleção diz respeito à vantagem que indivíduos mais aptos têm sobre os demais indivíduos; quanto maior for essa vantagem, maior diz-se que é a pressão de seleção.

Por exemplo, suponha uma população $P = (I_1, I_2, I_3, I_4)$ composta por quatro indivíduos cujos valores de aptidão são, respectivamente, 0.2, 0.6, 0.1 e 0.4. Sendo I_2 o indivíduo mais apto, deve-se esperar que ele seja amostrado com maior probabilidade do que os demais. Na Tabela 4.1 são dadas as probabilidades de seleção quando três diferentes operadores de seleção, s_1 , s_2 e s_3 são aplicados nessa população. Repare como o operador s_1 privilegia pouco os indivíduos mais aptos em relação aos demais, ao passo que o operador s_3 privilegia bastante o indivíduo I_2 . Pode-se dizer, portanto, que a pressão de seleção de s_1 é inferior à pressão de s_2 que, por sua vez, é inferior à de s_3 .

Tabela 4.1: Três operadores de seleção aplicados a uma população, exemplificando três diferentes cenários de pressão de seleção.

Indivíduo	Aptidão	Probabilidade de seleção		
		s_1	s_2	s_3
I_1	0.2	24,6%	18,7%	4,9%
I_2	0.6	26,2%	43,8%	77,8%
I_3	0.1	24,2%	6,2%	0,7%
I_4	0.4	25,0%	31,3%	16,6%

A pressão de seleção é um compromisso delicado no algoritmo evolutivo. Quando a pressão é muito baixa, a informação da aptidão é pouco utilizada e o algoritmo se assemelha a uma busca cega, como é o caso do operador s_1 exemplificado na Tabela 4.1. Quando a pressão é muito alta, por outro lado, o algoritmo se assemelha ao *hill climbing*, pois a cada instante t o algoritmo procura selecionar apenas as melhores soluções de $P(t)$. Tal situação é exemplificada pelo operador s_3 na mesma tabela.

Infelizmente, mensurar ou definir o valor da pressão de seleção objetivamente é tipicamente difícil e, em alguns casos, provavelmente impossível. Apesar disso, conhecer o operador de seleção utilizado e a forma como esse operador contribui para a pressão de seleção permite a escolha de operadores e parâmetros de execução provavelmente adequados ao domínio de aplicação.

Nas próximas sub-seções, alguns operadores de seleção serão abordados. Nas Seções 4.7 a 4.9 neste capítulo, outros elementos importantes do algoritmo evolutivo serão abordados.

4.6.1 A Roda da Roleta

O operador da roda da roleta seleciona os indivíduos através de uma amostragem aleatória com repetição cuja distribuição de probabilidades tem dependência diretamente proporcional aos valores de aptidão dos indivíduos. Assim, se em uma população $P(t) = (I_1, I_2, \dots, I_n)$ a somatória dos valores de aptidão é $F_{total} = f_1 + f_2 + \dots + f_n$, então cada indivíduo I_i tem probabilidade $\frac{f_i}{F_{total}}$ de ser selecionado a cada passo da amostragem.

A abstração típica do operador da roda da roleta é uma roleta de cassino dividida em várias faixas de tamanhos proporcionais aos valores de aptidão dos indivíduos. Um indicador fixo aponta inicialmente para uma faixa qualquer. A aplicação do operador equivale a rodar a roleta, de modo que a faixa que estará alinhada com o indicador fixo quando a roleta parar de girar indica o indivíduo que deverá ser selecionado. Na Figura 4.2, a roleta é dividida em cinco faixas que representam cinco indivíduos (I_1 , I_2 , I_3 , I_4 e I_5) e exemplifica a relação direta entre probabilidade de seleção e valor de aptidão.

Figura 4.2: Método de seleção roda da roleta (Pila, 2007)

Implementações deste operador tipicamente pré-processam a população ordenando os indivíduos de acordo com seus valores de aptidão – com resultados idênticos se a população for ordenada de forma crescente ou decrescente. Então, o primeiro indivíduo de acordo com essa ordem recebe um valor de *aptidão acumulada* idêntico ao seu valor de aptidão, ao passo que os demais recebem um valor de aptidão acumulada idêntico à aptidão acumulada do indivíduo anterior acrescida de seu próprio valor de aptidão.

Uma vez definidos os valores de aptidão acumulada, a implementação sorteia um número pseudoaleatório no intervalo $[0, F_{total})$ e realiza uma busca sequencial na população ordenada. O primeiro indivíduo cujo valor de aptidão acumulada for superior ao número sorteado é selecionado e copiado para a população intermediária. Esse processo de sorteio e busca é repetido até que a população intermediária contenha o número de indivíduos desejado – o que normalmente corresponde ao tamanho da população $P(t)$.

O operador da roleta tem sido amplamente utilizado na literatura. Contudo, a pressão de seleção imposta varia bastante e é altamente dependente da distribuição de valores de aptidão na população (Deb, 2000). Por exemplo, quando um indivíduo possui um valor de aptidão muito elevado em relação aos demais indivíduos, a pressão de seleção tende a ser muito elevada. Por outro lado, se os indivíduos possuem valores de aptidão muito semelhantes, a pressão de seleção tende a ser baixa (Grefenstette, 2000). Em

domínios nos quais esse fenômeno é indesejado, outros operadores, como o operador de seleção por *ranking*, podem ser utilizados.

4.6.2 Seleção por *ranking*

O operador de seleção por *ranking* pode ser encarado como uma variação do operador da roda da roleta, porém com uma etapa de pré-processamento dos valores de aptidão que permite um certo controle da pressão de seleção.

Assim como no operador da roleta, na aplicação deste operador os indivíduos são tipicamente ordenados de forma crescente ou decrescente – desta vez, entretanto, podendo levar a resultados diferentes, dependendo das especificidades do operador aplicado. Essa ordenação é utilizada na construção de um *ranking* e, para cada indivíduo um valor de *rank* é atribuído. Para propósito de comunicação, assuma que o *ranking* construído segue as regras do “*ranking* posicional” descrito na Seção 3.4. Assim, indivíduos cujos *rank*s são inferiores aos demais são mais aptos. Esse valor de *rank* é então utilizado para calcular um valor de *rank* normalizado para cada indivíduo.

Diferentes implementações podem utilizar diferentes formas de normalização, com diferentes impactos na pressão de seleção. Uma vez calculados os *rank*s normalizados, o operador de seleção por *ranking* calcula, para cada indivíduo, um valor de *rank* acumulado e realiza uma amostragem quase idêntica à amostragem promovida pelo operador da roda da roleta, com a diferença de utilizar o valor de *rank* acumulado, não o valor de aptidão acumulada, na seleção do indivíduo.

A seguir, dois métodos de normalização são apresentados, a normalização linear e a normalização exponencial.

Normalização linear: o *rank* linear de um indivíduo cujo *rank* é r é dado pela Equação 4.1. Nessa equação, α e ω são parâmetros de entrada e n é o tamanho da população.

$$Lin(r, \alpha, \omega) = \alpha + (\omega - \alpha) \times \left(\frac{n - r}{n - 1} \right) \quad (4.1)$$

Os valores α e ω influenciam na pressão de seleção da seguinte forma: quanto maior o valor de $\delta = \omega - \alpha$, maior tende a ser a pressão de seleção. Por outro lado, quanto menor o valor de α , menor tende a ser a influência de δ na pressão de seleção, de modo que a pressão de seleção tende a ser mais dependente da distribuição de aptidão.

Normalização exponencial: o *rank* exponencial de um indivíduo cujo *rank* é r é dado pela Equação 4.2. Nessa equação, $b \in [0, 1]$ é um parâmetro de entrada.

$$Exp(r, b) = b^{r-1} \quad (4.2)$$

Quanto mais b tende a zero, maior é a pressão de seleção. Com $b = 0$, $Exp(1, b)$ é indefinido, porém, como $Exp(r, 0) \rightarrow 1$ quando $r \rightarrow 0$, pode-se avaliar $Exp(1, 0) = 1$ e $Exp(r, 0) = 0$ para qualquer valor de *rank* estritamente positivo. Em outras palavras, com $b = 0$, a pressão de seleção é máxima e apenas os indivíduos que ocupam a primeira posição do *ranking* podem ser selecionados. Com $base = 1$, $Exp(r, b) = 1$ para todo valor positivo de r , de modo que a pressão de seleção se torna inexistente e a busca é cega.

É curioso observar que, se $b > 1$, um fenômeno que poderia ser encarado como uma espécie de “pressão de seleção inversa” ocorre: os indivíduos menos aptos tendem a ser selecionados com maior probabilidade. Essa “pressão inversa” é tão maior quanto maior for o valor de b .

O operador de seleção por *ranking* pode ser particularmente interessante quando a função de avaliação utilizada não define uma medida ou quando a proporção de valores de medida não reflete uma proporção igual ou semelhante em qualidade de solução. Por exemplo, é interessante aplicar o operador de seleção por *ranking* quando a própria função de avaliação é um *ranking* proveniente de uma simulação computacional que meramente estabelece uma ordem parcial entre os indivíduos. Isso porque a pressão de seleção imposta por este operador não depende de escala. Pelo mesmo motivo, este também pode ser interessante quando a função de avaliação possui crescimento assintótico elevado.

Para fins de eficiência, é interessante que o usuário experimente diferentes parâmetros e diferentes normalizações com o operador de seleção por *ranking*. Afinal, uma vez que a pressão de seleção dificilmente pode ser quantificada, não é possível prever de forma eficiente quais parâmetros poderão construir um cenário no qual a pressão de seleção seja adequada.

4.6.3 O Torneio

A seleção por torneio é um método razoavelmente simples, mas que permite bom controle da pressão de seleção. O primeiro passo consiste em amostrar um conjunto de indivíduos da população, organizando um *torneio*. Então, o indivíduo mais apto dentre os indivíduos participantes do torneio é declarado *campeão* e selecionado. O processo é repetido até que a população intermediária contenha o número de indivíduos desejado.

Apesar de sua simplicidade, o método de seleção por torneio é bastante flexível. Diferentes implementações podem utilizar diferentes especificidades para o método de amostragem, para o número de indivíduos participantes do torneio e para a forma

como o indivíduo campeão é escolhido. Tais especificidades podem influenciar a pressão de seleção de diferentes formas:

Amostragem: a amostragem pode ser feita com ou sem repetição dos indivíduos da população. A não-repetição garante que um mesmo indivíduo não será amostrado duas vezes para o torneio, mas não garante que vários indivíduos distintos, porém que codificam a mesma solução, sejam amostrados. Apesar disso, é consideravelmente simples implementar mecanismos para assegurar que indivíduos que codificam as mesmas soluções não participem de um mesmo torneio – *e.g.*, pode-se impedir um indivíduo de entrar no torneio se já houver outro com código genético idêntico.

A amostragem é habitualmente realizada de forma aleatória, com distribuição de probabilidades uniforme, porém aplicações mais específicas podem facilmente aplicar outras formas de distribuição, as quais podem ou não utilizar a informação de aptidão dos indivíduos. Entretanto, a escolha de uma distribuição diferente da uniforme representa um viés, o qual pode afetar a pressão de seleção de forma dependente da distribuição escolhida. Por exemplo, uma distribuição de probabilidades que selecione para o torneio indivíduos com menor aptidão pode aliviar a pressão de seleção, ao passo que selecionar para o torneio indivíduos com maior aptidão pode aumentar a pressão.

Tamanho do torneio: o número de indivíduos participantes do torneio determina o *tamanho* do torneio. O tamanho do torneio influencia diretamente a pressão de seleção. Quanto maior o torneio, maior a probabilidade de que os indivíduos mais aptos da população sejam amostrados, o que significa que a probabilidade de indivíduos menos aptos serem amostrados em torneios nos quais podem ser declarados campeões é reduzida. Portanto, quanto maior o torneio, maior a pressão de seleção.

Valores típicos de tamanho de torneio variam na faixa de 3% a 6% do tamanho da população. É comum que o tamanho do torneio seja fixado na primeira iteração e não seja alterado durante todo o processo de otimização, porém é possível que algoritmos mais robustos ajustem o torneio durante a execução para ajustar a pressão de seleção dinamicamente.

Declaração do campeão: tipicamente, o campeão do torneio é o indivíduo participante do torneio com maior valor de aptidão. Isso significa que a pressão de seleção tende a diminuir conforme o algoritmo progride e a distribuição de valores de aptidão entre os indivíduos se torna mais homogênea.

Em geral, implementações de algoritmo evolutivo onde esse fenômeno é uma preocupação tendem a não utilizar o operador de seleção por torneio. Entretanto, o torneio pode ser modificado de várias formas com o objetivo de reduzir ou eliminar esse fenômeno. Por exemplo, pode-se “desqualificar”, de forma aleatória,

indivíduos do torneio que sejam muito parecidos com outros indivíduos. Dependendo do domínio de aplicação, a utilização de tais medidas pode tanto produzir o resultado desejado e aliviar a pressão de seleção quanto aumentar ainda mais a pressão de seleção, prejudicando o algoritmo. Deve-se verificar isoladamente para cada caso, portanto, se há necessidade ou benefício em modificar o operador.

Assim como o operador de seleção por *ranking*, este operador é de grande interesse em situações nas quais a função de avaliação não define uma medida ou quando a medida definida pela função de avaliação não reflete a proporção de qualidade de soluções codificadas pelos indivíduos. Por exemplo, quando a função de avaliação é um *ranking* ou uma medida cujo nível inicial não é o zero – *e.g.*, temperatura em graus Celsius.

4.7 Operador de Crossover

No algoritmo genético, o *operador de crossover* simula o papel que a reprodução exerce na natureza. O operador de *crossover* modifica um número de indivíduos, tipicamente dois, construindo novos indivíduos rearranjando as informações dos indivíduos originais. De acordo com a teoria dos blocos de construção, o operador de *crossover*, aliado à função de avaliação e ao operador de seleção, é o mecanismo responsável por combinar os blocos de construção presentes na população para formar indivíduos mais aptos (Carvalho et al., 2003).

O operador de *crossover* é tipicamente aplicado imediatamente após o operador de seleção e atua sobre a população intermediária $P'(t)$. Para cada par de indivíduos na população intermediária, o operador de *crossover* produz um par de indivíduos, chamado *offspring*, o qual é inserido em uma segunda população intermediária $P''(t)$. A população intermediária $P''(t)$ fica disponível para o algoritmo evolutivo realizar novas modificações nos indivíduos ou copiar os indivíduos para a população da iteração seguinte, $P(t + 1)$.

O operador de *crossover* é aplicado de forma probabilística, podendo ou não modificar um par de indivíduos. A probabilidade de que o operador de *crossover* modifique um par de indivíduos é chamada *taxa de crossover* e define aproximadamente a proporção de pares de indivíduos que serão modificados pelo operador. Os pares que não são modificados pelo operador são copiados para a população intermediária $P''(t)$ no lugar do *offspring*.

Diferentes operadores de *crossover* atuam de diferentes formas nos indivíduos. Um exemplo bastante recorrente é o *crossover* de um ponto, ilustrado na Figura 4.3. O *crossover* de um ponto alinha os dados que codificam dois indivíduos e estabelece um ponto de corte comum a ambos, construindo dois novos indivíduos da permutação

dos dados separados pelo ponto de corte. Na Figura 4.3, I_1 e I_2 representam a informação codificada em dois indivíduos pertencentes a uma população temporária $P'(t)$. O ponto de corte é ilustrado pela reta vertical em I_1 e I_2 . Observe a *offspring*, representada por O_1 e O_2 , gerada pela troca do material genético dos indivíduos.

Figura 4.3: Ilustração do *crossover* de um ponto. I_1 e I_2 são representações abstratas de dois indivíduos quaisquer. O_1 e O_2 são representações abstratas das *offsprings* produzidas pelo operador.

O *crossover* de um ponto é bastante utilizado na literatura, sendo também comum a abstração deste operador para o *crossover* de n -pontos. O funcionamento do *crossover* de n -pontos é bastante semelhante ao *crossover* de um ponto. Porém, em vez de definir um único ponto de corte, são definidos n pontos de cortes.

É importante ressaltar que o operador de *crossover* não é capaz de introduzir informação nova na população, pois o operador de *crossover* apenas rearranja os blocos de construção presentes na população para formar novos indivíduos. Além disso, quanto maior a pressão de seleção, maior é a tendência de que o operador de *crossover* replique os blocos de construção contidos nos indivíduos mais aptos de uma população, os quais não necessariamente são blocos de construção que codificam soluções ótimas. Isso significa que o operador de *crossover* causa uma redução na *diversidade* da população. Eventualmente, a diversidade pode se tornar tão reduzida e os blocos de construção, de tão pequena variedade, que o algoritmo se torna incapaz de produzir indivíduos mais aptos do que aqueles presentes na população corrente. Em termos de otimização, isso significa que todos os indivíduos da população encontram-se em pontos de máximo do espaço de busca ou muito próximos de pontos de máximo e, portanto, que o algoritmo evolutivo alcançou a convergência.

A convergência do algoritmo evolutivo é importante, porém deve-se ter cautela para evitar que a convergência não ocorra antes que o algoritmo seja capaz de encontrar um ou mais indivíduos ótimos, o que caracterizaria uma situação de convergência prematura. Existem diversas formas de evitar a convergência prematura do algoritmo evolutivo, incluindo a especificação de uma taxa de *crossover* adequada, a utilização de um método de seleção com pressão de seleção apropriada e a aplicação de mecanismos para assegurar diversidade dos blocos de construção, como o operador de mutação e alguns mecanismos de manutenção de diversidade – Seção 4.9.

Operadores de *crossover* voltados para regras de conhecimento requerem manipulação dos indivíduos de forma consistente com as condições e os atributos dos dados. Na

Seção 5.3.1 será explicado como indivíduos são codificados neste trabalho. Em seguida, na Seção 5.3.3, os operadores de *crossover* implementados para a execução deste trabalho serão explicados.

4.8 Operador de Mutação

O *operador de mutação* desempenha, no algoritmo evolutivo, um papel semelhante ao fenômeno genético de mutação. Enquanto o operador de *crossover* é utilizado com a finalidade de replicar os indivíduos mais aptos ou os blocos de construção pertinentes aos indivíduos mais aptos, o operador de mutação é utilizado com a finalidade de inserir informação inédita na população (Michalewicz, 1997).

Assim como o operador de *crossover*, o operador de mutação é aplicado de forma probabilística. A probabilidade de que o operador de mutação seja aplicado em um indivíduo qualquer é chamada *taxa de mutação*. Um indivíduo $I_j \in P''(t)$ qualquer selecionado para mutação é modificado isoladamente dos demais e mantém-se na população intermediária $P''(t)$. No algoritmo evolutivo típico, os indivíduos da população intermediária $P''(t)$ são movidos para a população da iteração seguinte $P(t+1)$ após a aplicação do operador de mutação.

Em geral, os operadores de mutação atuam selecionando aleatoriamente um indivíduo e um “trecho” do conteúdo genético do indivíduo, tipicamente com distribuição de probabilidades uniforme. A definição exata de “trecho” depende das especificidades do operador e da codificação do indivíduo – *e.g.* no caso de codificação binária, um “trecho” pode ser um bit; no caso de representação *Michigan*, um “trecho” pode ser a cabeça da regra. Em seguida, o operador de alguma forma modifica esse “trecho”. Novamente, a modificação do “trecho” depende da codificação do indivíduo – *e.g.* no caso da codificação binária, o bit pode ter o seu valor negado; no caso da codificação de regras, a cabeça pode ser modificada para prever uma classe diferente da atual. A operação de mutação é genericamente ilustrada na Figura 4.4.

Figura 4.4: Ilustração de um operador de mutação. I_1 é uma representação abstrata do indivíduo original com um trecho do código genético selecionado. O_1 é uma representação abstrata do *offspring* produzido por uma modificação aleatória realizada no indivíduo original.

O operador de mutação possui um compromisso delicado entre inserir informação na população e modificar indivíduos previamente selecionados. Analisando sob a ótica da hipótese dos blocos de construção, a mutação de um indivíduo, ao mesmo tempo que insere blocos de construção possivelmente inéditos, modifica blocos de construção previamente selecionados pelo operador de seleção. Isso significa que, ainda que o operador de mutação seja importante em guiar o algoritmo evolutivo

em espaços não-explorados, ele pode “destruir” blocos de construção presentes nos indivíduos selecionados nos passos anteriores da iteração.

Por essa razão, é importante que a taxa de mutação seja escolhida adequadamente, pois, se a taxa de mutação for muito elevada, os blocos de construção selecionados pela função de seleção serão frequentemente modificados, o que pode prejudicar a capacidade do algoritmo evolutivo em encontrar e sustentar blocos de construção eficientes. Por outro lado, se a taxa de mutação for muito reduzida, então a inserção de novos blocos de construção será limitada e o algoritmo terá dificuldade em explorar o espaço de busca adequadamente.

4.9 Outras Considerações

Além dos elementos apresentados nas seções anteriores, outras considerações importantes para o algoritmo evolutivo incluem *critério de parada*, *convergência*, *diversidade* e *elitismo*.

Critério de parada: algoritmos evolutivos são processos iterativos de otimização. Em cada iteração t , o conjunto de indivíduos $P(t)$ representa um conjunto de soluções que atende com certo grau de satisfatoriedade o problema de otimização a ser resolvido. É esperado que, a cada iteração $P(t)$, $P(t + 1)$, $P(t + 2)$, $\dots$, o conjunto de soluções representadas pelos indivíduos seja cada vez mais eficiente em satisfazer o problema de otimização.

É visível que, em algum instante t , o algoritmo evolutivo deve ser interrompido e as soluções representadas pelos indivíduos da população $P(t)$ devem ser aceitas como um conjunto de soluções “boas o bastante” para resolver o problema de otimização. A situação ideal ocorre quando o algoritmo é interrompido após convergência para um ou mais pontos de máximo global. Entretanto, identificar a convergência pode ser difícil, especialmente quando o algoritmo evolutivo é utilizado para solucionar problemas de otimização multiobjetivo.

Quando detectar a convergência não é possível, abordagens típicas incluem definir um número máximo de iterações t_{max} , tal que quando $t \leq t_{max}$ a execução é interrompida. Outra abordagem comum consiste em definir um valor de aptidão máximo, f_{max} , tal que quando um indivíduo I_i qualquer alcançar o valor de aptidão f_{max} , a execução é interrompida.

Não existe forma canônica para o critério de parada. Para cada domínio de aplicação, é necessário que o usuário estabeleça um critério específico considerando as características do domínio de aplicação e, possivelmente, alguns testes preliminares. É também recomendável que sempre seja definido um número máximo de iterações, por ser um critério simples e que garantidamente é satisfeito durante a execução.

Diversidade: Manter um grau mínimo de diversidade na população diminui a probabilidade de convergência prematura, ao mesmo tempo em que assegura que o algoritmo será capaz de encontrar soluções variadas para problemas de otimização multimodal ou multiobjetivo.

O algoritmo evolutivo multiobjetivo NSGA-II possui mecanismos de diversidade por padrão. Outra forma, bastante direta, de buscar a manutenção da diversidade consiste em definir uma medida de distância entre os indivíduos e utilizar essa medida como um objetivo de otimização extra. Com isso os próprios mecanismos do algoritmo evolutivo padrão podem ser utilizados para “otimizar” a diversidade da população ao mesmo tempo que outros objetivos são otimizados.

Elitismo: elitismo é utilizado para garantir que os indivíduos mais aptos da população serão selecionados, em contraste com o mecanismo típico de seleção no qual indivíduos mais aptos possuem maior probabilidade, mas não garantia, de serem selecionados. O elitismo também pode ser utilizado para garantir que os indivíduos mais aptos serão copiados para a iteração seguinte sem sujeição às alterações efetuadas pelos operadores de *crossover* e de mutação.

A forma mais direta de aplicação de elitismo é utilizada em problemas de otimização simples. Em uma iteração t qualquer, após a etapa de avaliação, o indivíduo mais apto da população I_w é identificado. Após as etapas de seleção e modificação – *crossover* e mutação – o indivíduo I_w é copiado diretamente da população $P(t)$ para a população $P(t + 1)$, substituindo um indivíduo qualquer. Com isso é estabelecida a garantia de que o indivíduo I_w não será descartado nos passos de seleção e modificação. Outra aplicação consiste em, após a construção da população intermediária $P''(t)$, repetir a etapa de seleção, desta vez considerando tanto os indivíduos de $P(t)$ quanto os indivíduos de $P''(t)$. Os indivíduos selecionados por essa segunda etapa de seleção são copiados diretamente para a população seguinte $P(t + 1)$. Isso diminui a probabilidade de que indivíduos mais aptos da população original $P(t)$ sejam substituídos por indivíduos menos aptos resultantes das etapas de modificação da população.

O elitismo pode ser visto como um mecanismo para aumentar a probabilidade de que modificações positivas feitas pelo algoritmo evolutivo não sejam descartadas durante as etapas de modificação e seleção. Entretanto, uma consequência do elitismo é o aumento da pressão de seleção, o que pode se refletir em redução da diversidade da população. É recomendável que o usuário experimente diferentes abordagens elitistas e não-elitistas para determinar qual situação melhor se adequa ao seu domínio de aplicação.

4.10 Considerações Finais

O presente capítulo realizou uma revisão bibliográfica sobre computação evolutiva e algoritmos evolutivos, apresentando uma visão geral sobre os seus principais componentes e conceitos. No próximo capítulo, diversos desses componentes e conceitos são revistos, agora, instanciados em uma aplicação de algoritmos evolutivos na indução de regras de conhecimento, a ECLE. Portanto, conceitos como codificação de indivíduos, *crossover* e mutação são discutidos em detalhes para fornecer ao leitor uma noção de como esses componentes de um algoritmo genético foram implementados na ECLE, e os possíveis impactos que a implementação desses componentes possui na pesquisa realizada nessa Dissertação.

Ambiente Computacional

5.1 Considerações Iniciais

Nos capítulos anteriores foram discutidos os conceitos de aprendizado de máquina, otimização multiobjetivo e computação evolutiva necessários para a compreensão deste trabalho. Neste capítulo será descrito o ambiente computacional utilizado para execução e avaliação das propostas desta Dissertação. Tal ambiente é essencialmente a biblioteca de classes e ambiente de construção de regras ECLE – *Evolutionary Computing Learning Environment*. Também é utilizada na execução deste trabalho a biblioteca de classes DOL – Discover Object Library –, parte integrante do projeto DISCOVER.

A ECLE é uma biblioteca de classes e simultaneamente um ambiente de construção de regras de conhecimento. Ela emprega um algoritmo de computação evolutiva multi-objetivo ou simples-objetivo que pode utilizar um ou mais critérios de qualidade para guiar um processo de computação evolutiva na construção de regras de conhecimento que atendam a esses critérios. Tais critérios podem ser medidas de qualidade ou de interesse de regras, como aquelas apresentadas na Seção 2.5, bem como quaisquer outros mecanismos de avaliação de regras de conhecimento que possam ser implementados como uma função de avaliação do algoritmo evolutivo.

Apesar de ser voltada especificamente para a construção de regras de conhecimento, a ECLE não possui rotinas específicas para a manipulação de exemplos e avaliação de regras de conhecimento. Em vez disso, ela emprega interfaces que permitem a comunicação da ECLE com outros sistemas responsáveis por essas etapas da construção de regras, cabendo à ECLE a manipulação dos indivíduos do algoritmo genético que representam essas regras. Neste trabalho de Mestrado e em outros trabalhos que envolveram a ECLE, as etapas de avaliação de regras de conhecimento são realizadas pela biblioteca de classes DOL, com a qual a ECLE se comunica por meio de uma interface.

Este capítulo tem três propósitos. O primeiro é explicar como as propostas deste trabalho são implementadas como funcionalidades da ECLE. O segundo é ambientar o leitor, fornecendo uma visão geral sobre a DOL e da estrutura da ECLE. O terceiro é fornecer ao leitor uma noção de quais partes da ECLE foram desenvolvidas em projetos anteriores a este trabalho e quais partes são contribuições deste trabalho. Sendo assim, o restante deste capítulo está organizado como se segue: na Seção 5.2 são apresentados o ambiente DISCOVER e a biblioteca DOL. Na Seção 5.3 é apresentada uma visão geral sobre a organização dos diferentes componentes da ECLE. Indivíduos, função de avaliação, operadores de *crossover* e operadores de mutação são abordados nas Seções 5.3.1, 5.3.2, 5.3.3 e 5.3.4, respectivamente. Na Seção 5.4 é explicado como o algoritmo evolutivo é utilizado pela ECLE para a construção de regras de conhecimento.

5.2 A Biblioteca de Classes DOL

O projeto DISCOVER (Baranauskas e Batista, 2000) foi concebido por pesquisadores do Laboratório de Inteligência Computacional – LABIC – com o objetivo de reduzir problemas de implementação, integração e replicação de ferramentas e algoritmos desenvolvidos pelos membros do laboratório. O DISCOVER pode ser entendido como um conjunto de métodos para manipular dados e conhecimento por meio de sintaxes padrão (Batista e Monard, 2003b; Batista, 2001; Prati et al., 2001) para a representação de dados e do conhecimento induzido por diferentes algoritmos de aprendizado, bem como por bibliotecas que oferecem um conjunto de funcionalidades básicas para tais manipulações.

O DISCOVER conta com diferentes ambientes, tais como o ambiente Discover Learning Environment – DLE –, composto pela biblioteca de classes Discover Object Library – DOL –, e o ambiente para gerenciamento de experimentos SNIFFER, que foram desenvolvidos para as diferentes tarefas de pré-processamento de dados no formato atributo-valor e análise dos classificadores induzidos por algoritmos de aprendizado utilizados no processo de Mineração (Batista, 2003).

A biblioteca DOL – Discover Object Library – é uma biblioteca orientada a objetos baseada em padrões de projeto (Gamma et al., 1995). As classes dessa biblioteca implementam as tarefas de manipulação e gerenciamento de dados mais comuns em pré-processamento, tais como gerenciamento de diferentes sintaxes de arquivos de dados e atributos, amostragens, métodos de *resampling*, estatísticas descritivas, normalizações de dados, entre outros (Batista e Monard, 2003b). O objetivo da DOL é ser uma biblioteca de métodos de pré-processamento e dar suporte à criação de novos métodos de pré-processamento de dados.

Neste trabalho, a DOL é utilizada para carregar os exemplos utilizados no processo de indução de regras, bem como fornecer informações de domínio essenciais para a

ECLE, como a natureza dos atributos – qualitativo ou quantitativo –, o valor médio amostral de atributos quantitativos, o valor de moda de atributos qualitativos etc. A DOL também é utilizada para realizar a avaliação das regras de conhecimento representadas pelos indivíduos. A cada iteração do algoritmo genético, a ECLE exporta para a DOL um conjunto de regras de conhecimento e recebe como resposta as matrizes de contingência das regras. Com base nos valores das matrizes de contingência, a ECLE pode calcular diversas medidas de qualidade de regra – atualmente, todas as apresentadas nas Seções 2.5.1 e 2.5.2 – que são utilizadas como função de avaliação do algoritmo evolutivo.

A DOL também é utilizada para construir um conjunto inicial de regras de conhecimento que é utilizado pela ECLE como população inicial do algoritmo evolutivo. Utilizando indutores clássicos, como C4.5, C4.5R e CN2, a DOL estabelece um *pool* inicial de blocos de construção potencialmente melhor, para fins de otimização, do que uma população gerada aleatoriamente, ainda que a ECLE permita a utilização de população inicial aleatória ou de regras quaisquer definidas pelo usuário.

Finalmente, a DOL é utilizada na avaliação dos métodos de otimização implementados. Técnicas de *resampling* implementadas pela DOL são utilizadas para realizar particionamento dos conjuntos de dados, conforme explicado na Seção 6.3.

5.3 A Biblioteca de Classes ECLE

A ECLE – *Evolutionary Computing Learning Environment* – é uma biblioteca de classes e um ambiente computacional que utiliza computação evolutiva simples-objetivo e multiobjetivo para construir regras de conhecimento utilizando critérios configuráveis (Pila et al., 2006a; Giusti et al., 2006). Cabe ressaltar que, embora seja voltada especificamente para a construção de regras de conhecimento, a ECLE não possui rotinas específicas para manipulação de exemplos e avaliação de regras de conhecimento. Em vez disso, a ECLE emprega uma interface que utiliza outros ambientes computacionais dedicados à avaliação de regras de conhecimento e manipulação de exemplos. Neste trabalho e em trabalhos anteriores nos quais a ECLE foi empregada, essa interface utiliza a biblioteca de classes DOL para a manipulação de exemplos e avaliação de regras de conhecimento.

A arquitetura da ECLE pode ser descrita como um conjunto de sete pacotes de classes, os quais se inter-relacionam e se relacionam com a DOL conforme mostrado na Figura 5.1. Uma seta direcionada de um pacote *A* a um pacote *B* indica que uma ou mais classes em *A* utiliza métodos de uma ou mais classes em *B*.

A ECLE pode ser subdividida em três conjuntos de pacotes: o conjunto dos pacotes de operadores genéticos, o conjunto unário formado pelo pacote *Application* e o conjunto unário formado pelo pacote *Interface*. É interessante observar aqui que, embora a ECLE tenha sofrido diversas adaptações para que fosse possível adotar uma

Figura 5.1: Arquitetura de pacotes da ECLE.

estratégia de otimização multiobjetivo, grande parte das modificações realizadas nesta Dissertação ficam concentradas no pacote Application.

O conjunto dos pacotes de operadores genéticos é formado pelos pacotes Individuals, Fitness, Selection, Crossover e Mutation oferece métodos e estruturas de dados para manipulação de indivíduos, cálculo de aptidão dos indivíduos com base nos valores da matriz de contingência e aplicação de operadores de seleção, *crossover* e mutação, bem como análise de dominância, extração de fronteira de Pareto, construção de *rankings*, construção de regras de conhecimento aleatórias etc. Todos os pacotes de operadores genéticos seguem a restrição de não utilizarem classes e funções dos pacotes restantes da ECLE. Assim, esses pacotes poderiam ser utilizados como uma biblioteca de classes independente do restante das classes da ECLE.

O pacote Application contém classes que facilitam a implementação de diferentes modalidades de algoritmos de computação evolutiva. As classes deste pacote facilitam a instanciação de operadores genéticos e oferecem métodos para controlar o fluxo do algoritmo. Diferentes implementações de algoritmos de computação evolutiva podem ser implementados com a utilização das classes deste pacote, incluindo o algoritmo evolutivo típico, o algoritmo evolutivo multiobjetivo VEGA e o algoritmo evolutivo multiobjetivo NSGA-II.

Finalmente, o pacote Interface contém classes que realizam a interface da ECLE com a biblioteca DOL. Dentre outras, este pacote contém rotinas que instanciam um objeto da classe RuleSet, fornecida pela DOL para a manipulação de regras de conhecimento, a partir de um objeto Population, fornecido pelo pacote Individuals para a manipulação de indivíduos que representam essas regras. O pacote Interface também contém rotinas que atualizam as matrizes de contingência de objetos da classe Individual com base nas regras contidas em um objeto da classe RuleSet.

Uma vez que as classes dos pacotes de operadores genéticos não utilizam ou referenciam os demais pacotes da ECLE, esses pacotes podem ser utilizados isoladamente dos demais. Ficará, nesse caso, a encargo da aplicação controlar o fluxo de chamadas

para implementar os passos do algoritmo evolutivo e também para suprir informações sobre as regras de conhecimento representadas pelos indivíduos.

Nas subseções a seguir as principais funcionalidades oferecidas pelos diferentes pacotes da ECLE são explicadas. Nem todas as classes contidas nos pacotes são apresentadas nesta Dissertação. Detalhes de implementação são omitidos por questão de brevidade.

5.3.1 Indivíduos

Os indivíduos da ECLE são codificados para representar regras de conhecimento compatíveis com o formato $\mathcal{PBM}$ (Prati et al., 2001). O formato $\mathcal{PBM}$ é um formato padrão do projeto DISCOVER para descrição de regras de classificação, sendo que o DISCOVER provê um conjunto de ferramentas para converter arquivos de regras de diferentes indutores, tais como C4.5, CN2, RIPPER etc para o formato $\mathcal{PBM}$, bem como uma coleção de métodos capazes de manipular regras nesse formato. Entre as funcionalidades que são mais tipicamente utilizadas, o DISCOVER permite verificar quais exemplos são cobertos por uma determinada regra e qual é a classificação fornecida pela regra para esse exemplo. Uma regra de conhecimento só faz sentido quando associada a um domínio de aplicação, do qual tipicamente infere-se uma distribuição de características a partir de um conjunto de dados bem-amostrado. Assim, regras representadas no formato $\mathcal{PBM}$ são compostas por corpo, cabeça coerentes com o domínio de aplicação e duas matrizes de contingência associadas a um conjunto de exemplos do mesmo domínio.

Para cada regra avaliada, duas matrizes de contingência são associadas, uma referente aos exemplos do conjunto de dados cujos valores são definidos para todos os atributos e outra referente aos exemplos do conjunto de dados que contêm valores desconhecidos em um ou mais atributos. Dois vetores representando as frequências relativas f_{hb} , $f_{\bar{h}b}$, $f_{h\bar{b}}$ e $f_{\bar{h}\bar{b}}$, nessa ordem, mais um valor inteiro que indica o número absoluto de exemplos com valores conhecidos ou desconhecidos, são associados à regra. O primeiro vetor apresentado tipicamente refere-se aos exemplos sem valores desconhecidos, ao passo que o segundo refere-se a exemplos com valores desconhecidos – ainda assim, uma interrogação precede o segundo vetor para reforçar a distinção.

Considere novamente o conjunto de dados *tênis* apresentado na Tabela 2.2 na Seção 2.4.1. Para conveniência do leitor, o conjunto de dados está reproduzido na Tabela 5.1 a seguir.

Um exemplo de regra de conhecimento descrita em um formato semelhante ao $\mathcal{PBM}$ foi dado na Seção 2.4.2 – pg. 34. Na Figura 5.2 a seguir um novo exemplo é mostrado, desta vez incluindo as matrizes de contingência. Note que os valores das matrizes é dado em frequências relativas aos 14 exemplos do conjunto de dados. Note também

Tabela 5.1: Reprodução da Tabela 2.2, apresentando o conjunto de dados tnis.

	Aspecto	Temperatura	Umidade	Vento	JogarT�nis
E_1	Aberto	Alta	Elevada	Fraco	N�oJogar
E_2	Aberto	Alta	Elevada	Forte	N�oJogar
E_3	Nublado	Alta	Elevada	Fraco	Jogar
E_4	Chuvoso	M�dia	Elevada	Fraco	Jogar
E_5	Chuvoso	Baixa	Normal	Fraco	Jogar
E_6	Chuvoso	Baixa	Normal	Forte	N�oJogar
E_7	Nublado	Baixa	Normal	Forte	Jogar
E_8	Aberto	M�dia	Elevada	Fraco	N�oJogar
E_9	Aberto	Baixa	Normal	Fraco	Jogar
E_{10}	Chuvoso	M�dia	Normal	Fraco	Jogar
E_{11}	Aberto	M�dia	Normal	Forte	Jogar
E_{12}	Nublado	M�dia	Elevada	Forte	Jogar
E_{13}	Nublado	Alta	Normal	Fraco	Jogar
E_{14}	Chuvoso	M�dia	Elevada	Forte	N�oJogar

que, como no existem exemplos com valores desconhecidos no conjunto, a matriz de contingncia correspondente aos exemplos com valores desconhecidos possui apenas frequncias nulas.

```

if Aspecto = Chuvoso
  and Vento = Fraco
  then class = Jogar
[0.21, 0.00, 0.36, 0.43, 14]  ?[0.00, 0.00, 0.00, 0.00, 0]

```

Figura 5.2: Exemplo de regra de conhecimento no formato $\mathcal{PBM}$.

O formato $\mathcal{PBM}$ importante para a ECLE, pois permite a opo de induzir um conjunto inicial de regras de conhecimento que podem ser utilizadas para construir a populao inicial do algoritmo evolutivo. Utilizando indutores clssicos, como C4.5, C4.5R e CN2, e convertendo os classificadores simblicos construdos por cada indutor para o formato $\mathcal{PBM}$ possvel gerar um *pool* inicial de blocos de construo que provavelmente sero melhores, para fins de otimizao, do que aqueles contidos em uma populao gerada aleatoriamente – como tipicamente realizado em aplicaes de computao evolutiva. Apesar disso, a ECLE pode construir regras aleatrias, bem como permitir que seja utilizadas regras especificadas pelo usurio ou que tenham quaisquer outras origens, contanto que estas sejam representadas no formato $\mathcal{PBM}$. Esse processo ilustrado na Figura 5.3.

Como mencionado anteriormente, a ECLE no manipula regras de conhecimento, mas indivduos que representam essas regras. Por esse motivo a ECLE utiliza no o formato $\mathcal{PBM}$, mas uma codificao prpria capaz de representar regras de conhecimento compatveis com o formato $\mathcal{PBM}$. O indivduo codificado por um vetor que contm o corpo da regra, uma cadeia de caracteres que identifica a classe predita e duas matrizes 2×2 que representam as matrizes de contingncia, conforme pode ser

Figura 5.3: Conceituação geral de uso típico da ECLE, ilustrando a definição do *pool* inicial de regras de conhecimento a partir de fontes quaisquer, convertidas para o formato *PBM*, e aplicação do algoritmo evolutivo para construir o conjunto final de regras de conhecimento com propriedades específicas ou critérios de interesse do usuário. (Giusti et al., 2006)

verificado na Figura 5.4 a seguir. Nessa codificação o vetor que representa o corpo da regra possui tamanho arbitrário, porém fixo, de condições.

No vetor que codifica o corpo da regra, cada condição é um par de restrições relacionadas a um dos atributos do conjunto de dados. Todo indivíduo possui número de condições idêntico ao número de atributos do conjunto de dados e, para todos os indivíduos de uma população, qualquer i -ésimo elemento do vetor é correspondente ao mesmo atributo. Por sua vez, cada restrição é uma tripla que contém um *bit* de atividade, um operador de condição e um valor limitante. O *bit* de atividade de uma restrição qualquer recebe o valor 1 e é dito *ligado* se estiver associado a um complexo do corpo da regra. Caso contrário, o *bit* indicativo de atividade recebe o valor 0 e é dito *desligado*. Sempre que uma restrição possui *bit* de atividade desligado, os valores associados ao operador e ao limitante da restrição são ignorados pela ECLE. Note que, como o vetor de condições possui tamanho fixo para todos os indivíduos, é de se esperar que muitas restrições não estejam associadas a complexos e que, portanto, tenham seus *bits* de atividade desligados.

Figura 5.4: Codificação da regra de conhecimento

Por questões de homogeneidade, a mesma estrutura de dados é utilizada para representar condições referentes a atributos qualitativos e atributos quantitativos. Isso facilita a implementação dos operadores de *crossover* e mutação, mas exige que as restrições sejam tratadas de acordo quando se quer obter o complexo associado ou quando do complexo quer-se obter a restrição que o codifica.

Por padrão, todas as restrições possuem o *bit* de atividade desligado. Dependendo dos complexos presentes no corpo da regra que o indivíduo representa, alguns *bits* de atividade são ligados. No caso de condições referentes a atributos qualitativos, o *bit* de atividade da segunda restrição é sempre mantido desligado, ao passo que o primeiro

bit é ligado se, e somente se, um complexo do tipo $\alpha_{trib} = S$ estiver presente no corpo da regra representada. Nesse caso, o valor do operador da restrição é definido tal que representa o operador $=$ e o valor do limitante é a cadeia de caracteres S . No caso de condições referentes a atributos quantitativos, as restrições são analisadas separadamente:

- Se a regra representada pelo indivíduo contém um complexo do tipo $\alpha_{trib} > X$ ou do tipo $\alpha_{trib} \geq X$, então liga-se o *bit* de atividade da primeira restrição da condição correspondente – limitante inferior. O valor do operador é definido tal que representa $>$ ou $\geq$ e o valor limitante é X ;
- Se a regra representada pelo indivíduo contém um complexo do tipo $\alpha_{trib} < X$ ou do tipo $\alpha_{trib} \leq X$, então liga-se o *bit* de atividade da segunda restrição da condição correspondente – limitante superior. O valor do operador é definido tal que representa $<$ ou $\leq$ e o valor limitante é X ;

Para melhor ilustrar como essa codificação é feita, considere a regra de conhecimento ilustrada na Figura 5.5. Mesmo sem conhecer o conjunto de dados que possibilitou a construção dessa regra, pode-se intuitivamente verificar que ela se refere a um domínio de concessão de crédito, isto é, ilustra conhecimento relacionado a se aprovar ou recusar um empréstimo com base em certas características do requerente, as quais são representadas pelos atributos do conjunto de dados. Claro, trata-se de um exemplo bastante simplificado, porém seu propósito é ilustrar a codificação da regra em indivíduo.

```

if salário (R$ mil)  $\geq$  2
  and salário (R$ mil)  $\leq$  4,5
  and casado = sim
  then empréstimo = sim
[0.80, 0.00, 0.00, 0.20, 20] ?[0.00, 0.00, 0.00, 0.00, 0]

```

Figura 5.5: Exemplo de regra *PBM* de um domínio de concessão de crédito.

O indivíduo que codifica a regra apresentada na Figura 5.5 é ilustrado na Figura 5.6. Observe que o *bit* de atividade está desligado para a segunda restrição da condição referente ao atributo *casado*, já que se trata de um atributo de natureza qualitativa. Como existem dois complexos associados ao atributo *salário* (R\$ mil), o indivíduo contém duas restrições com *bits* de atividades ligados.

Na ECLE existem duas classes utilizadas para implementação e manipulação de indivíduos. A codificação de cada indivíduo com condições, restrições e matrizes de contingência é realizada pela classe *Individual*. Um conjunto de indivíduos que forma uma população independente é implementado pela classe contêiner *Population*. As duas classes pertencem ao pacote *Individuals*. A conversão de indivíduos em regras e a alimentação das matrizes de contingência dos indivíduos após a avaliação das regras pelo DOL é realizada pela classe *Conversion*, contida no pacote *Interface*.

Figura 5.6: Exemplo de codificação da regra de conhecimento

5.3.2 Função de Avaliação e Operador de Seleção

Conforme explicado nas Seções 4.5 e 4.6, a função de avaliação e o operador de seleção simulam no algoritmo evolutivo a competição que ocorre entre os indivíduos na natureza. Nesta, de acordo com a teoria da seleção natural, os indivíduos mais aptos tendem a sobreviver enquanto os menos aptos tendem a padecer, de modo que os primeiros têm mais chances de transmitir suas características. No algoritmo evolutivo, a função de seleção avalia os indivíduos e define aqueles que são mais e menos aptos. O operador de seleção seleciona os indivíduos da população de forma aleatória, porém dependente da informação de aptidão dos indivíduos, de modo que os indivíduos que retratam soluções melhores tendem a permanecer, ao passo que os demais tendem a ser descartados.

Na ECLE, o pacote Fitness contém classes que implementam diferentes funções de avaliação. Para cada uma das medidas de qualidade dentre as apresentadas nas Seções 2.5.1 e 2.5.2 existe uma classe que recebe como entrada um objeto da classe *Individual* e retorna um valor que reflete a qualidade da regra representada pelo indivíduo codificado pelo objeto. Por exemplo, a classe *PrecisionMeasure* implementa a medida de precisão, enquanto a classe *CosineMeasure* implementa a medida Cosseno. Já a classe *CompositeMeasure* é uma classe contêiner que guarda diferentes objetos de medidas associados a diferentes valores de pesos, calculando a aptidão do indivíduo por meio de uma média aritmética ou harmônica ponderada das medidas.

A composição de *rankings* também é implementada como uma classe no pacote Fitness. Cada método de composição de *ranking* é implementado por uma classe que estende a classe *CompositeRankingMeasure*, que é por sua vez uma extensão da classe *CompositeMeasure*. A classe *CompositeRankingMeasure* herda as rotinas para manipular diferentes objetos de medidas e acrescenta rotinas para construir *rankings* dos indivíduos de acordo com as diferentes medidas associadas. Classes que implementam os métodos de composição de *rankings* – e.g. *CondorcetRanking* – possuem métodos que recebem como entrada um objeto da classe *Individual* e retornam um valor de aptidão referente ao *rank* composto da regra representada pelo indivíduo codificado no objeto.

Os métodos de seleção, por sua vez, são implementados por classes do pacote Selection. Cada classe desse pacote – e.g., *TournamentSelection* — possui métodos que

recebem como entrada um objeto da classe *Population* e retornam um novo objeto da classe *Population*. O objeto retornado possui cópias dos objetos que codificam os indivíduos amostrados pelo método de seleção.

5.3.3 Operadores de Crossover

Na ECLE, os operadores de *crossover* são implementados por diferentes classes contidas no pacote *Crossover*. O pacote possui uma classe abstrata base, *AbstractCrossover*, a qual é estendida para definir classes que implementam diferentes operadores de *crossover*. Atualmente, três operadores estão implementados.

Todas as classes implementam um método virtual *cross* que recebe dois indivíduos pais e constrói dois indivíduos *offspring* que são retornados para a função chamadora. Os indivíduos originais não são alterados.

A ECLE também permite que seja aplicada uma composição de operadores de *crossover* por meio da classe *CompositeCrossover*. Refere-se a essa operação como *crossover* composto. O *crossover* composto aplica diferentes operadores de *crossover*, em sequência, a um mesmo par de indivíduos, sendo cada operador aplicado com probabilidade igual à taxa de *crossover*.

Nas próximas subseções, são apresentados os três operadores de *crossover* implementados atualmente na ECLE.

Crossover Estrutural

O *crossover* estrutural, implementado pela classe *StructuralCrossover*, implementa o *crossover* de um ponto – Seção 4.7 – para indivíduos que representam regras de conhecimento. Esse operador define um *ponto de corte* nesse atributo e promove uma troca de condições entre os indivíduos, conforme exemplificado na Figura 5.7. Em Figura 5.7 (a), dois indivíduos pertencentes à população temporária $P'(t)$ são primeiramente alinhados e um ponto de corte comum a ambos é aleatoriamente selecionado com distribuição de probabilidades uniforme. As *offsprings* são construídas com as condições à esquerda do ponto de corte de um pai. O passo seguinte, ilustrado em 5.7 (b), consiste em combinar as condições que antecedem o ponto de corte em uma das regras com as condições que sucedem o ponto de corte na outra regra e vice-versa.

No contexto do algoritmo genético, o *crossover* estrutural manipula blocos de construção que correspondem às condições das regras de conhecimento. Dessa forma este operador promove a busca por uma regra de conhecimento que contenha as melhores condições presentes na população. Entretanto, este operador não é capaz de promover alterações nos cortes feitos por cada condição individualmente. Para essa finalidade são utilizados os operadores de *crossover* apresentados nas seções seguintes.

Figura 5.7: Exemplo de aplicação do *crossover* estrutural.

Crossover de Atributo

O operador de *crossover* de atributo modifica condições das regras de conhecimento relacionadas a um atributo. A aplicação deste operador consiste em selecionar aleatoriamente, com distribuição de probabilidades uniforme, um dos atributos presentes no conjunto de dados e combinar as condições possivelmente especificadas pelas regras aos valores desse atributo. A *offspring* resultante da aplicação deste operador é aproximadamente idêntica aos indivíduos originais, com exceção do atributo selecionado pelo operador.

Quando o domínio consiste em regras de conhecimento que obedecem a representação de regras *PBM*, os indivíduos sempre codificam regras que possuem uma ou nenhuma condição associada a um atributo qualitativo qualquer. Assim, no caso de atributos qualitativos, a aplicação do operador de *crossover* de atributo consiste simplesmente em “trocar” as condições codificadas pelos indivíduos originais.

Na mesma situação de regras *PBM*, atributos quantitativos podem possuir zero, uma ou duas condições associadas. Isto é, um atributo a_i quantitativo qualquer pode não ser utilizado no corpo de uma regra de conhecimento ou pode ser utilizado de três formas distintas, como exemplificado a seguir:

- IF $a_i \geq x$ THEN
- IF $a_i < y$ THEN
- IF $a_i > x$ AND $a_i \leq y$ THEN

Dessa forma, especificar condições para um atributo quantitativo equivale a definir um intervalo. Quando apenas uma condição relacionada ao atributo é apresentada pela regra, esse intervalo é aberto ao infinito em um dos extremos – *e.g.* $[x, \infty)$ e

$(-\infty, y)$. Quando duas condições são utilizadas pela regra, esse intervalo é aberto ou fechado em valores nos dois extremos, dependendo dos operadores utilizados pelas condições – *e.g.* $(x, y]$. Caso a regra codificada pelo indivíduo não utilize o atributo em nenhuma condição, então pode-se dizer que ela codifica o intervalo $(-\infty, \infty)$.

Tendo em mente a notação de intervalos, pode-se dizer que a aplicação deste operador em atributos quantitativos consiste em promover entre os indivíduos uma “troca” de um dos extremos dos intervalos codificados, como pode ser observado na Figuras 5.8. Nessa figura, o atributo *salário* foi selecionado para aplicação do operador de *crossover*. Em 5.8 (a) pode-se observar dois indivíduos da população temporária $P'(t)$, o primeiro codificando o intervalo completamente fechado $[2, 4.5]$ e o segundo codificando o intervalo $[6, \infty)$. Em 5.8 (b) são ilustrados os indivíduos resultantes da aplicação do operador. Neste caso, os indivíduos tiveram seus intervalos inferiores trocados, resultando indivíduos que codificam os intervalos $[6, 4.5]$ e $[2, \infty)$. Caso a troca fosse realizada nos intervalos superiores, os indivíduos resultantes codificariam os intervalos $[2, \infty)$ e $[6, 4.5]$.

Figura 5.8: Exemplo de aplicação do *crossover* de atributo.

Um problema inerente a este operador é que o intervalo resultante pode ser demasiadamente estreito ou mesmo inválido – *e.g.*, na Figura 5.8, um dos indivíduos contém o intervalo $[6, 4.5]$ em suas condições. Esta situação ocorre sempre que a interseção dos intervalos originais é vazia e produz indivíduos cujas regras codificadas são incapazes de cobrir quaisquer exemplos. A tendência associada a este fenômeno é a construção de indivíduos que serão severamente “punidos” pela função de avaliação, não sendo selecionados na iteração seguinte. Como a etapa de *crossover* do algoritmo evolutivo tipicamente transporta apenas a *offspring* para formar a população intermediária $P''(t)$, pode-se dizer que essa ocorrência causa a “morte” de um dos indivíduos originais da população intermediária $P'(t)$.

Esse fenômeno é frequentemente considerado indesejável, porém facilmente contornável. Entre as soluções possíveis, pode-se trocar os extremos de um intervalo

inválido ou multiplicar os extremos do intervalo por -1 . Também é possível evitar que o operador de *crossover* seja aplicado em atributos quantitativos cujos intervalos contenham interseção vazia, impedindo assim a formação de indivíduos inválidos.

Crossover Local

O *crossover* local modifica os valores das condições das regras codificadas pelos indivíduos. Sua aplicação consiste em selecionar um atributo aleatoriamente, com distribuição de probabilidades uniforme, e promover uma alteração nos valores de uma das condições associadas ao atributo, caso haja.

Um exemplo de aplicação do *crossover* local é dado na Figura 5.9. Em 5.9 (a), dois indivíduos originais são extraídos da população temporária $P'(t)$ e uma única condição referente a um atributo é selecionada. A *offspring* é então gerada pela cópia dos indivíduos originais, acrescidas das modificações à condição selecionada, conforme mostrado em 5.9 (b).

Figura 5.9: Exemplo de aplicação do *crossover* local.

Diferentes implementações do operador de *crossover* local podem utilizar diferentes mecanismos para compor os valores das condições. No caso de atributos qualitativos, o mais comum é que as condições sejam simplesmente trocadas entre os indivíduos, de forma que o operador de *crossover* local tem o mesmo efeito que o operador de *crossover* de atributo para atributos qualitativos. No caso de atributos quantitativos, por sua vez, é comum a aplicação do operador *alpha-blend* ou **BLX**- α (Herrera et al., 2002).

Os valores dos indivíduos originais x e y são utilizados como parâmetros de entrada na Equação 5.1. Os valores x' e y' resultantes são utilizados como os valores de condição para os indivíduos da *offspring*. A constante α ($\alpha \in [0, 1]$), chamada *fator de mistura - blend* – é definida pelo usuário e serve para “calibrar” a intensidade das

modificações que o operador de *crossover* aplicará aos indivíduos. Com $\alpha = 0$, os valores de x e y não são alterados. Com $\alpha = 1$, os valores de x e y são alternados. Valores intermediários de α produzem composições intermediárias.

$$\begin{cases} x' &= x - \alpha \times (x - y) \\ y' &= y + \alpha \times (x - y) \end{cases} \quad (5.1)$$

Assim como ocorre com o *crossover* de atributo, o *crossover* local também pode produzir indivíduos cujas regras codificadas sejam incapazes de cobrir exemplos. De forma semelhante também, é condição necessária que os intervalos representados pelas condições apresentem interseção vazia. Entretanto, a interseção vazia não é uma condição suficiente para a ocorrência deste fenômeno. Como os valores finais dependem do fator de mistura α , a ocorrência deste fenômeno também dependerá do fator de mistura, porém é possível verificar que, quanto mais $\alpha \rightarrow 1$, maior a probabilidade de ocorrência de intervalos inválidos.

5.3.4 Operadores de Mutação

Na ECLE, os operadores de mutação são implementados por diferentes classes contidas no pacote *Mutation*. O pacote possui uma classe abstrata base, *MutationAbstract*, a qual é estendida para definir classes que implementam diferentes operadores de mutação. Atualmente, dois operadores estão implementados.

Todas as classes implementam um método virtual *mutate* que recebe um indivíduo e constrói um indivíduo derivado que é retornado para a função chamadora. O indivíduo original não é alterado.

A ECLE também permite que seja aplicada uma composição de operadores de mutação por meio da classe *CompositeMutation*. Refere-se a essa operação como mutação composta. A mutação composta aplica diferentes operadores de mutação, em sequência, a um mesmo indivíduo, sendo cada operador aplicado com probabilidade igual à taxa de mutação.

Nas próximas subseções, são apresentados os dois operadores de mutação implementados atualmente na ECLE.

Mutação Estrutural

O operador de *mutação estrutural* modifica a estrutura da regra codificada pelo indivíduo, acrescentando ou removendo condições. Em um primeiro passo, este operador seleciona aleatoriamente, com distribuição de probabilidades uniforme, um atributo do conjunto de dados. Caso o atributo selecionado seja de natureza qualitativa, o operador adiciona ou remove uma condição referente a esse atributo, dependendo da existência prévia dessa condição. Caso o atributo selecionado seja de natureza

quantitativa, o operador seleciona entre um limitante superior ou inferior e adiciona ou remove a condição relacionada a esse limitante, dependendo da existência prévia dessa condição.

Um exemplo da aplicação do operador de mutação estrutural é dado na Figura 5.10. Em 5.10 (a) é mostrada a regra codificada pelo indivíduo original, a qual possui três condições: $\text{salário} \geq 2$, $\text{salário} \leq 4,5$ e $\text{casado} = \text{sim}$. O operador de mutação estrutural é aplicado, tendo sido sorteado o atributo *casado*, o qual é de natureza qualitativa. Uma vez que a regra codificada pelo indivíduo original já possui uma condição referente a esse atributo, a modificação consiste em remover essa condição. O indivíduo resultante é mostrado em 5.10 (b).

Figura 5.10: Exemplo de mutação estrutural.

É interessante analisar, sob a ótica da hipótese dos blocos de construção, como este operador pode ser capaz de acrescentar informação inédita à população. Para o operador de mutação estrutural, os blocos de construção são porções de dados que codificam condições das regras. Para cada atributo qualitativo do conjunto de dados, o indivíduo contém um bloco de construção associado. Para cada atributo quantitativo, o indivíduo contém dois blocos de construção associados – uma possivelmente definindo um limitante inferior e outro possivelmente definindo um limitante superior. É importante ressaltar que, mesmo quando a regra codificada pelo indivíduo não possui nenhuma condição relacionada a um atributo a_i qualquer, o indivíduo ainda assim possui em seu código genético um ou dois blocos de construção relacionados ao atributo, os quais podem estar contido no indivíduo de forma explícita (e.g., o indivíduo pode ter uma variável indicando ausência da condição na regra codificada) ou de forma implícita.

Quando o operador é aplicado e a condição selecionada já está presente na regra codificada pelo indivíduo, o papel do operador consiste em converter o bloco de construção de “condição presente” para “condição ausente”. Essa troca pode representar informação de novos blocos de construção, porém é improvável, pois essa informação de ausência apenas será inédita se todos os indivíduos presentes na população possuírem blocos de presença relacionados à condição selecionada.

Quando o operador é aplicado e a condição selecionada não está presente na regra codificada pelo indivíduo, entretanto, o operador atua ativamente na inserção de informação “original”. Tal noção de “original” está relacionada à ideia de que o bloco

de construção é formulado sem nenhuma informação do indivíduo original ou de outros indivíduos da população. Sendo assim é importante estabelecer um critério para que o operador de mutação estrutural seja capaz de formular novas condições para as regras de conhecimento codificadas pelos indivíduos. Um exemplo dessa situação é dado na Figura 5.11. Em 5.11 (a), a regra codificada pelo indivíduo original é mostrada. O operador de mutação estrutural seleciona o atributo salário e insere a condição $\text{salário} \leq 4,5$.

Figura 5.11: Exemplo de mutação estrutural com inserção de condição

O operador de mutação estrutural deve ser capaz de inserir informação nova à população, porém com o cuidado de não “matar” indivíduos previamente selecionados. Isso porque, se a mutação for realizada de forma descontrolada, é possível que condições incompatíveis com outras condições da regra ou fora do domínio estabelecido pelos exemplos do conjunto de dados sejam inseridas. Isso pode causar grande prejuízo ao valor de aptidão do indivíduo, tornando improvável sua seleção em iterações posteriores do algoritmo evolutivo.

Mutação Local

O operador de *mutação local* modifica o valor da condição da regra codificada pelo indivíduo. Assim como o operador de mutação estrutural, este operador é aplicado em um atributo selecionado aleatoriamente com distribuição de probabilidades uniforme. No caso de atributos qualitativos, a aplicação deste operador consiste apenas em modificar o valor do atributo especificado pela condição relacionada. No caso de atributos quantitativos, a aplicação deste operador depende da escolha aleatória de um limitante e, em seguida, da alteração do valor associado à condição referente ao limitante selecionado.

Um exemplo da aplicação deste operador é dado na Figura 5.12. Em 5.12 (a) é mostrada a regra codificada por um único indivíduo, a qual possui três condições: $\text{salário} \geq 2$, $\text{salário} \leq 4,5$ e $\text{casado} = \text{sim}$. A modificação é feita no valor do limitante superior da condição relacionada ao atributo salário, alterando o valor codificado pelo indivíduo de 4.5 para 6.2. A regra codificada pelo indivíduo resultante é apresentada em 5.12 (b).

Figura 5.12: Exemplo de aplicação do operador de mutação local.

Não existe forma canônica para a aplicação do operador de mutação local. Várias abordagens podem ser adotadas, tanto para atributos qualitativos quanto para atributos quantitativos, no sentido de definir como o operador modificará os valores ou os operadores das condições codificadas pelos indivíduos. Algumas sugestões incluem:

Atributos qualitativos: cada atributo qualitativo a_i possui um conjunto de valores factíveis $A_i = \{a_{i1}, a_{i2}, \dots, a_{i\xi}\}$ que podem estar presentes nos exemplos do conjunto de dados. Uma regra de conhecimento no formato $\mathcal{PBM}$ só pode possuir uma ou nenhuma única condição relacionada a um atributo a_i qualquer, a qual sempre tem a forma $a_i = a_{ij}$. A aplicação do operador de mutação local para atributos qualitativos consiste, portanto, em alterar o valor corrente a_{ij} especificado pela condição para um novo valor $a'_{ik} \in A_i$.

Uma abordagem direta consiste em selecionar a'_{ik} de forma aleatória, com distribuição de probabilidades uniforme entre todos os valores factíveis de A_i . Para evitar que o valor corrente da condição a_{ik} seja selecionado novamente como o novo valor da condição, pode-se fazer a seleção sobre o conjunto $A_i/\{a_{ik}\}$ ou pode-se repetir a seleção caso o novo valor seja idêntico ao valor corrente.

Abordagens mais específicas podem utilizar distribuições de probabilidades dependentes do valor corrente do atributo. Por exemplo, se houver o auxílio de um especialista de domínio na aplicação do algoritmo evolutivo, o especialista pode indicar que certos valores para o atributo são improváveis, de modo que o usuário pode escolher aumentar – caso esteja em busca de regras inesperadas – ou diminuir – caso tenha preferência por classificação precisa – a probabilidade de seleção desse valor pelo operador de mutação. Também é possível que o operador verifique a distribuição de valores para o atributo entre as regras codificadas pelos indivíduos da população temporária $P''(t)$, de modo a privilegiar ou prejudicar a seleção de valores muito frequentes entre as regras – novamente, a escolha depende da intenção do usuário.

Atributos quantitativos: um atributo quantitativo a_i possui um conjunto de valores nos exemplos que podem ser agrupados no conjunto $V_i = \{x_{1i}, x_{2i}, \dots, x_{Ni}\}$, onde

x_{ji} é o valor do atributo a_i no j -ésimo exemplo. Ordenando os valores de V_i pode-se obter quatro estatísticas do conjunto de dados:

1. O Valor mínimo do atributo a_i no conjunto de dados, $\min(x_i)$;
2. O valor máximo do atributo a_i no conjunto de dados, $\max(x_i)$;
3. A diferença mínima entre dois valores quaisquer, $\alpha = \min(|x_{ji} - x_{(j+1)i}|)$;
4. A diferença máxima entre dois valores sequenciais $\omega = \max(|x_{ji} - x_{(j+1)i}|)$.

A aplicação do operador de mutação local em um atributo quantitativo consiste em realizar um “desvio” no valor corrente v_i – um incremento ou decremento. Esse “desvio” pode ser um valor fixo δ ou um valor aleatório selecionado com qualquer distribuição de probabilidades no intervalo $[\delta_{\min}, \delta_{\max}]$. Em ambas as situações, é possível que δ , $\delta_{\min}$ e $\delta_{\max}$ sejam especificados pelo usuário ou inferidos pelo algoritmo evolutivo.

No caso de inferência pelo algoritmo evolutivo, é interessante explorar os valores α e ω como limites do intervalo $[\delta_{\min}, \delta_{\max}]$. Por exemplo, é possível utilizar $\delta_{\min} = 0$ e $\delta_{\max} = \alpha$ ou $\delta_{\max} = \omega$. Ou ainda $\delta_{\min} = \alpha$ e $\delta_{\max} = \omega$. Isso permite que uma aplicação mais genérica do algoritmo evolutivo para construção de regras de conhecimento e classificadores.

No exemplo da Figura 5.12, a operação de mutação é realizada com $\alpha = \min(x_i)$. No hipotético conjunto de dados relacionado, tem-se que $\min(x_i) = 0,7$. A aplicação do operador nesse exemplo consiste, portanto, na seleção aleatória do atributo salário, seguida da seleção aleatória do limitante superior, seguida da seleção aleatória de incremento. O valor $\alpha = 1,7$ é então acrescido ao valor original da condição codificada pela regra, produzindo a regra ilustrada em 5.12 (b).

É importante ressaltar que o operador de mutação local desempenha um importante papel na inserção de informação na população – ou, no contexto de algoritmos genéticos, na geração de blocos de construção inéditos. A aplicação eficiente deste operador depende de uma boa escolha do novo valor da condição, especialmente no caso de atributos quantitativos. Como todo operador de mutação, este operador deve aplicar modificações significativas o bastante para que os novos indivíduos acrescentem informação inédita à população, mas não tão significativas a ponto de modificar exageradamente indivíduos previamente selecionados pelo operador de seleção.

5.4 Algoritmos Evolutivos Implementados na ECLE

Os pacotes de operadores genéticos Individuals, Fitness, Selection, Crossover e Mutation oferecem uma gama de métodos e estruturas de dados para a implementação de

diferentes algoritmos de computação evolutiva para a construção de regras de conhecimento. Por exemplo, para implementar o algoritmo evolutivo típico, descrito pelo Algoritmo 2.1 apresentado na Seção 4.3, pode-se utilizar a classe `Population` para instanciar uma população inicial de indivíduos. A função de avaliação pode ser definida instanciando-se uma classe do pacote `Fitness` e operadores de seleção, *crossover* e mutação podem ser facilmente definidos instanciando-se classes dos respectivos pacotes. O trabalho da aplicação consiste então em controlar o fluxo de mensagens e fornecer às classes dos pacotes de operadores genéticos informações sobre o domínio dos dados e os valores das matrizes de contingência de cada regra representada por cada indivíduo – o que pode ser facilmente obtido com a utilização do pacote `Interface`.

O pacote `Application` contém classes que utilizam as classes restantes da `ECLE` e algumas classes da `DOL` para implementar diferentes algoritmos evolutivos para construção de regras de conhecimento. Atualmente, existem quatro classes neste pacote que implementam algoritmos evolutivos, a saber, `TypicalEA`, `MultiObjectiveEA`, `VEGA` e `NSGA2`. Além disso, existe a classe `AbstractEA`.

Todas as classes do pacote `Application` foram escritas ao longo da execução deste projeto de Mestrado. Mesmo a classe `TypicalEA`, a qual implementa um algoritmo evolutivo típico, sem nenhuma relação necessária com composição de *rankings* ou outros métodos de otimização multiobjetivo, foi re-escrita a partir da modificação de classes que deixaram de existir, visando uma melhor organização das classes.

O pacote `Application` é estruturado de forma que as classes nele contida possam ser utilizadas tanto quanto um indutor quanto uma interface de programação. Utilizadas como uma interface de programação, o usuário pode instanciar as classes conforme achar conveniente – respeitando, é claro, as exigências de pré-condições e pós-condições de classes e métodos – para implementar o algoritmo evolutivo que lhe parecer mais adequado. Quando a `ECLE` é utilizada como um indutor, entretanto, ela segue aproximadamente os seguintes passos:

1. Um objeto da classe `Controller` do pacote `Application` é instanciado. Um objeto da classe `Configure` é instanciado e este objeto recebe um arquivo de configuração de experimento;
2. O objeto da classe `Configure` interpreta o arquivo de configuração e instancia todos os objetos necessários para a execução do algoritmo evolutivo, incluindo um objeto de uma das classes que implementa o algoritmo evolutivo, *e.g.*, `MultiObjectiveEA`. O conjunto de dados utilizado no processo de indução é carregado por classes da `DOL` e, utilizando classes do pacote `Interface`, o *pool* inicial de regras representadas no formato $\mathcal{PBM}$ é carregado e uma população inicial de indivíduos é construída;
3. O objeto da classe `Controller` utiliza uma das classes que implementa o algoritmo evolutivo definido pelo arquivo de configuração para executar, iterativamente, o

algoritmo evolutivo para construção de regras de conhecimento. Essa classe oferece métodos para detecção do critério de parada e também para a produção de relatórios, os quais são exibidos para o usuário pelo objeto da classe Controller para que o usuário acompanhe o processo de otimização iteração-a-iteração;

4. Quando o critério de parada é detectado pelo objeto da classe que implementa o algoritmo evolutivo, a classe Controller salva em arquivo um relatório final, bem como um arquivo de regras no formato *PBM* contendo as regras representadas pelos indivíduos da população final.

A distinção mais significativa entre diferentes execuções da ECLE como um indutor diz respeito ao algoritmo evolutivo aplicado. Atualmente, existem quatro algoritmos evolutivos implementados na ECLE: o algoritmo evolutivo típico para resolução de problemas de otimização simples-objetivo, o algoritmo evolutivo simples-objetivo para resolução de problemas de otimização multiobjetivo utilizando composição de *rankings* e os algoritmos evolutivos multiobjetivo para resolução de problemas de otimização multiobjetivo VEGA e NSGA-II. Destes, o algoritmo evolutivo típico para resolução de problemas de otimização simples-objetivo, implementado pela classe *TypicalEA*, é considerado legado e recebe pouca atenção neste trabalho, pois realiza construção de regras de conhecimento utilizando um único critério ou realiza composição de múltiplos critérios sem considerar a necessidade de apresentar diversas regras satisfatórias ao usuário. Os demais algoritmos evolutivos são explicados nas subseções a seguir.

5.4.1 Algoritmo Evolutivo Simples-Objetivo Baseado em *Rankings*

Conforme explicado na Seção 3.4, a utilização de *rankings* é uma estratégia que converte um problema de otimização multiobjetivo em um problema de otimização simples-objetivo. Métodos de composição de *rankings* são aplicados como funções de avaliação que substituem as funções objetivo originais, as quais passam a ser utilizadas pelo método de composição como variáveis de decisão. Assim, se um problema de otimização multiobjetivo possui inicialmente M funções objetivo, $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_M(\mathbf{x})$, o método de composição de *rankings* é uma função objetivo $F(f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_M(\mathbf{x}))$ que deve sempre ser minimizada.

Uma vez que os propósitos do método de otimização multiobjetivo são encontrar soluções eficientes e ao mesmo tempo fornecer uma gama diversa de soluções para o usuário, aos métodos de composição por *rankings* faz-se necessário o uso de um mecanismo específico para manutenção da diversidade da população. Existem diversas formas de se manter a diversidade. Uma delas é utilizar uma medida de diversidade como uma função adicional a ser otimizada. Por exemplo, considerando-se cada indivíduo como um ponto no espaço de objetivos, pode-se utilizar como métrica de diversidade uma função $f(\mathbf{x})$ que verifique a distância euclidiana total do ponto para

outros pontos da população nesse mesmo espaço. É possível ainda utilizar pesos para aumentar ou diminuir a importância da diversidade no processo de otimização.

A ECLE atualmente disponibiliza cinco métodos de composição de *rankings*, conforme explicado nas Seções 3.5 e 5.3.2. Para construir regras de conhecimento utilizando múltiplos critérios compostos por *rankings*, utiliza-se a classe MultiObjectiveEA e uma das seguintes classes para estabelecer o método de composição de *rankings* desejado: ArithmeticMeanRankingMeasure, GeometricMeanRankingMeasure, MedianRankingMeasure, CondorcetRankingMeasure ou ReciprocalRankingMeasure. Os critérios de otimização são definidos instanciando-se objetos de um conjunto de classes de medidas e associando-se os objetos instanciados ao objeto de composição de *rankings* previamente instanciado.

Independente da classe de composição de *rankings* utilizada, a ECLE atua aproximadamente da mesma forma: em cada iteração $t = 1, 2, \dots$, todos os N indivíduos pertencentes à população $P(t)$ são avaliados com relação a cada uma das funções objetivo previamente estabelecidas. Em seguida, os valores atribuídos a cada um dos indivíduos por cada função objetivo são utilizados para construir M *rankings* que são então compostos de acordo com o método de composição escolhido. Como resultado, cada indivíduo I_j de $P(t)$ recebe um valor de *rank* composto r_j , do qual se calcula a aptidão do indivíduo I_j como $f = N - r_j$. Esse valor de aptidão é utilizado pelo algoritmo nas etapas de seleção, *crossover* e mutação assim como ocorre no algoritmo evolutivo típico. Neste ponto então se faz uma distinção do algoritmo evolutivo simples-objetivo para otimização multiobjetivo do algoritmo evolutivo típico: no algoritmo típico, esse ciclo de avaliação, composição, seleção e modificação é repetido até que haja a convergência de um indivíduo para um ótimo – global, preferencialmente – ou outro critério de parada seja satisfeito. No algoritmo simples-objetivo utilizando *rankings*, o conceito de convergência é analisado não com relação a um único indivíduo, mas com relação à fronteira de Pareto que pode ser obtida da população corrente $P(t)$, conforme explicado na Seção 3.6.

5.4.2 Vector Evaluated Genetic Algorithm

O *Vector Evaluated Genetic Algorithm* – VEGA – foi uma das primeiras soluções baseadas em computação evolutiva para otimização de problemas multiobjetivos utilizando conceitos de dominância. Proposto por Schaffer (1984), o VEGA otimiza um vetor de objetivos no qual cada elemento do vetor representa uma função objetivo distinta. O VEGA é implementado na ECLE pela classe VEGA, do pacote Application.

O VEGA é uma simples extensão do algoritmo evolutivo típico para problemas de otimização simples-objetivo. Se o número de funções objetivo a serem otimizadas é M , a cada iteração o VEGA particiona a população inicial $P(t)$ em M subpopulações disjuntas $P_1(t), P_2(t), \dots, P_M(t)$ de forma aleatória. Todos os indivíduos são avaliados por uma única função objetivo, como no algoritmo evolutivo típico, porém em cada

partição é considerada uma função objetivo distinta. Deste modo, se as funções objetivos são $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_M(\mathbf{x})$, então os indivíduos da partição $P_1(t)$ são avaliados pela função $f_1(\mathbf{x})$, os indivíduos da partição $P_2(t)$ são avaliados pela função de avaliação $f_2(\mathbf{x})$ e assim por diante.

Após a etapa de avaliação, o operador de seleção é aplicado em cada partição separadamente, selecionando indivíduos para compor a população intermediária $P'(t)$. É importante que o operador de seleção seja aplicado isoladamente em cada partição, pois como as funções objetivos podem ser bastante distintas em termos de escala, distribuição e até mesmo objetivo – maximização ou minimização – os valores de aptidão dos indivíduos em partições distintas não são comparáveis. Definida a população intermediária $P'(t)$, o VEGA aplica os operadores de *crossover* e mutação como no algoritmo evolutivo típico, definindo a população seguinte $P(t+1)$ e reiniciando todo o processo na iteração seguinte. O ciclo continua até que a convergência seja alcançada ou outro critério de parada seja satisfeito.

Observe que o VEGA não possui nenhum mecanismo de manutenção de diversidade inerente. Assim como os métodos de composição de *rankings*, faz-se necessária a aplicação explícita de tal mecanismo, podendo ser definido um objetivo extra na definição do problema de otimização para permitir que o VEGA busque otimizar também a diversidade da população, além das funções originalmente definidas.

5.4.3 Elitist Non-Dominated Sorting Genetic Algorithm

Dado um conjunto qualquer de soluções no espaço de objetivos, é possível determinar o conjunto de soluções não-dominadas, denominado fronteira de Pareto. Se as soluções contidas na fronteira de Pareto forem temporariamente removidas do conjunto de soluções, pode-se verificar uma “segunda” fronteira de Pareto, composta pelas soluções não-dominadas, exceto por soluções presentes na “primeira” fronteira de Pareto. Repetindo-se esse procedimento até que não restem soluções, obtém-se uma ordenação por não-dominância – *non-dominated sorting*.

Como resultado da ordenação por não-dominância, obtém-se ω fronteiras hierárquicas $F_1, F_2, \dots, F_\omega$. Embora não seja um termo recorrente, neste texto diz-se que uma fronteira F_i qualquer é *superior* a uma fronteira F_j se $i < j$. Em contrapartida, toda fronteira F_j é *inferior* a F_i se $i \geq j$ – observe que toda fronteira é inferior a ela mesma. Assim, as ω fronteiras se configuram de forma tal que toda solução $S_{i\gamma}$ pertencente à fronteira F_i não é dominada por nenhuma solução contida em fronteiras inferiores e é dominada por pelo menos uma solução contida em pelo menos uma das fronteiras superiores – com exceção, é claro, das soluções $S_{1\gamma}$, $\gamma \in [1, |F_1|]$.

A ordenação por não-dominância é a base do algoritmo evolutivo multiobjetivo *Elitist Non-dominated Sorting Genetic Algorithm* – NSGA-II (Deb et al., 2000). O NSGA-II foi proposto por Kalyanmoy Deb que junto com N. Srinivas anteriormente propôs o *Non-*

dominated Sorting Genetic Algorithm (Goldberg, 1989) – NSGA. Entretanto, apesar da similaridade dos nomes e de autoria, os algoritmos NSGA e NSGA-II possuem poucas semelhanças (Deb, 2001).

Dada uma população $P(t)$ de tamanho N , a primeira etapa do processo de aplicação do NSGA-II consiste em, com o uso de um operador genético de seleção, construir uma população intermediária $P'(t)$ de $P(t)$ e submeter os indivíduos de P – doravante o parâmetro (t) será omitido – a operadores de *crossover*, originando assim uma sub-população Q composta por N *offsprings*. Os detalhes de aplicação do operador, incluindo valores de taxa de *crossover*, são livres. Em seguida, as populações P e Q são unidas, gerando uma nova subpopulação R , de tamanho $2 \times N$, que é então ordenada por critério de não-dominância.

Na etapa seguinte, considerando-se a ordenação por não-dominância de R , cada indivíduo $R_{ir} \in F_i$, com $i \in [1, \omega]$ e $r \in [1, |F_i|]$, recebe duas medidas: um *rank* de não-dominância e um valor de *crowding distance* local (Deb, 2001). O *rank* de não-dominância é simplesmente o índice i da fronteira F_i ao qual o indivíduo pertence, indicando o quanto o indivíduo se destaca dos demais em termos de eficiência de Pareto; quanto menor o *rank*, melhor. O *crowding distance* local, por sua vez, é uma estatística que estima a distância do indivíduo para outros indivíduos da mesma fronteira no espaço de objetivos. O propósito do *crowding distance* é estimar quão distinto um indivíduo é, considerando-se otimização dos objetivos, quando comparado com outros indivíduos semelhantes a ele em termos de eficiência de Pareto. Isto consiste um mecanismo interno do NSGA-II para buscar não apenas um conjunto de soluções próximo da fronteira Pareto-ótimo no espaço de objetivos, mas também uma fronteira o mais extensa possível.

Tendo-se separado os indivíduos em fronteiras de não-dominância e tendo-se mensurado o *rank* de não-dominância e o valor de *crowding distance* para todos os indivíduos da população, o NSGA-II realiza uma seleção elitista de indivíduos para compor uma nova população intermediária Z . A seleção é feita da seguinte forma: inicialmente, o algoritmo tenta selecionar todos os indivíduos da fronteira F_1 para Z . Se todos os indivíduos da fronteira “couberem” em Z , isto é, se $|F_1| \leq N$, então estes são copiados para Z – ou seja: $Z \leftarrow Z \cup F_1$. Esse processo é realizado repetidamente enquanto para as fronteiras $i = 2, 3, \dots, \omega$ enquanto a fronteira F_i “couber” em Z , isto é, enquanto $|Z| + |F_i| \leq N$.

Note que não é possível inserir todos os indivíduos de todas as fronteiras em Z , uma vez que a população R construída no início do procedimento possui $2 \times N$ indivíduos. Assim, eventualmente o algoritmo se depara com uma fronteira F_i que não “cabe” em Z , isto é, tal que $|Z| + |F_i| > N$. Quando isso ocorre, em vez de copiar todos os indivíduos de F_i para Z , o algoritmo copia os $N - |Z|$ indivíduos de F_i com maior valor de *crowding distance*.

Como resultado, a população intermediária Z contém N indivíduos com os menores valores possíveis de *rank* de não-dominância e, no caso dos indivíduos com maior *rank* dentre os indivíduos de Z , menor valor de *crowding distance*. Essa população é então submetida a um operador de seleção específico do NSGA-II chamado *crowded tournament selection*. O *crowded tournament selection* é semelhante ao operador de seleção por torneio, mas distingue-se deste em dois pontos. Primeiramente, o *crowded tournament selection* realiza sempre torneios com tamanho 2. Além disso, em vez de selecionar como vencedor do torneio o indivíduo com maior aptidão, seleciona como vencedor do torneio o indivíduo que:

1. possui *rank* de não-dominância estritamente menor que o *rank* de seu competidor;
2. possui *rank* de não-dominância igual ao de seu competidor, porém possui um valor mais elevado da medida *crowding distance*.

O *crowded tournament selection* é executado N vezes, selecionando um indivíduo de Z para a população intermediária em cada passo. Assim como ocorre com o operador de seleção por torneio “comum”, o *crowded tournament selection* amostra os indivíduos integrantes do torneio com distribuição uniforme e com reposição. Por essa razão, se nenhum dos dois critérios do *crowded tournament selection* for satisfeito, não é necessário efetuar desempate por sorteio. Após a aplicação do *crowded tournament selection* o algoritmo procede de forma semelhante ao algoritmo evolutivo típico, com a aplicação de *crossover*, mutação e a composição da população $P(t+1)$. O ciclo continua até que a convergência seja alcançada ou outro critério de parada seja satisfeito. Na ECLE, o NSGA-II é implementado pela classe NSGA2 do pacote Application.

5.5 Considerações Finais

Neste capítulo foi apresentada a biblioteca de classes e ambiente de construção de regras de conhecimento ECLE como parte principal do ambiente computacional utilizado no desenvolvimento deste trabalho. A estrutura geral da ECLE foi explicada e uma breve descrição de sua arquitetura foi fornecida.

É importante salientar que a investigação e implementação dos algoritmos VEGA e NSGA-II, explicados na Seção 5.4, constituem parte dos objetivos deste projeto de pesquisa, bem como a modificação da ECLE para utilização dos métodos de composição de *rankings* em um contexto de otimização multiobjetivo. Ambos os objetivos foram executados com sucesso.

Um terceiro e importante objetivo deste trabalho consiste da comparação da eficiência de métodos de composição de *rankings* contra a eficiência dos métodos de otimização baseados em computação evolutiva multiobjetivo. Para cumprir esse objetivo, a ECLE

foi utilizada para realização de diversos experimentos, os quais são discutidos no próximo capítulo desta Dissertação.

Avaliação Experimental

6.1 Considerações Iniciais

Como explicado no Capítulo 5, todas as propostas deste trabalho foram implementadas como funcionalidades da biblioteca de classes e ambiente de construção de regras de conhecimento ECLE. Foi também esclarecido que parte das funcionalidades atualmente existentes na ECLE haviam sido implementadas durante outros projetos de pesquisa dos quais o autor desta Dissertação tomou parte, ao passo que outras foram acrescentadas ou adaptadas durante a execução deste projeto.

Até o início das atividades deste projeto, a única abordagem utilizada pela ECLE para construir regras com base em critérios múltiplos consistia em converter um problema de otimização multiobjetivo – *i.e.*, construir regras que maximizem diversas medidas de qualidade potencialmente conflitantes – em um problema de otimização simples-objetivo por meio de composição de *rankings*. Tal conversão permite que o problema de otimização multiobjetivo seja solucionado com a aplicação de um algoritmo de computação evolutiva simples-objetivo e, como demonstrado por (Pila, 2007), é capaz de construir regras de conhecimento que atendam aos critérios estabelecidos pelo usuário. Apesar disso, trabalhos anteriores com a ECLE e métodos de otimização por composição de *rankings* levaram ao questionamento de que tal abordagem, ainda oferecendo bons resultados, poderia não ser capaz de fornecer os melhores resultados possíveis.

Dentre as propostas deste trabalho, incluem-se a investigação e implementação de métodos de computação evolutiva multiobjetivo e sua comparação com os métodos de computação evolutiva baseados em composição de *rankings* previamente existentes na ECLE. Dois métodos de computação evolutiva multiobjetivo, VEGA e NSGA-II, foram estudados e implementados como funcionalidades da ECLE, e diversos experimentos utilizando diferentes conjuntos de dados e diferentes cenários de aplicação

desses métodos foram executados e os resultados foram utilizados na comparação dos métodos.

A avaliação dos métodos de otimização multiobjetivo foi feita a partir da comparação das fronteiras de Pareto dos conjuntos de soluções construídos pelos métodos. As fronteiras foram comparadas tanto em termos de dominância ou proximidade com a fronteira Pareto-ótimo quanto em termos de diversidade das soluções. Essa análise permitiu concluir que os métodos de computação evolutiva multiobjetivo superam os métodos baseados em composição de *rankings*, o que também permite concluir que a hipótese deste trabalho de pesquisa é verdadeira.

O objetivo deste capítulo é esclarecer o método de avaliação experimental empregado e realizar uma discussão sobre os resultados obtidos. Na Seção 6.2 são apresentados os critérios considerados na comparação os diferentes métodos e as razões pelas quais esses critérios foram levantados. Nas Seções 6.2.1 e 6.2.2 são apresentadas as métricas utilizadas para a comparação dos métodos segundo os critérios estabelecidos, bem como as análises realizadas para a definição dessas métricas. Na Seção 6.3 os conjuntos de dados e os cenários de comparação utilizados são explicados. Na Seção 6.4 os resultados obtidos são apresentados e analisados. Finalmente, na Seção 6.5 são traçadas algumas considerações sobre os resultados obtidos e sobre a hipótese desta pesquisa.

6.2 Critérios de Comparação

Nos projetos de pesquisa anteriores envolvendo a ECLE a avaliação experimental consistia em considerar as regras de conhecimento representadas pelas populações iniciais e finais e comparar o ganho obtido da melhor regra inicial para a melhor regra final considerando as medidas de qualidade utilizadas. Por exemplo, supondo um cenário de teste envolvendo as medidas sensibilidade e precisão, no início do experimento era selecionado um dos indivíduos com melhor valor de aptidão (no caso simples-objetivo) ou melhor *rank* composto (no caso multiobjetivo) e seus valores nas medidas de sensibilidade – diga-se $Sen(R_1)$ – e precisão – diga-se $Pre(R_1)$ – eram comparados com os valores de sensibilidade e precisão do melhor indivíduo da população final selecionado pelo mesmo método – diga-se $Sen(R_2)$ e $Pre(R_2)$. As diferenças $Sen(R_2) - Sen(R_1)$ e $Pre(R_2) - Pre(R_1)$ eram utilizadas para comparar se houve “ganho” na utilização da ECLE. Os métodos de otimização que apresentassem melhor “ganho” eram considerados mais eficientes.

Com a modificação da ECLE para utilizar análise da eficiência de Pareto e considerar a fronteira de Pareto como método de se determinar quais indivíduos representam melhores regras, em detrimento do *rank* composto, foi necessário estabelecer uma nova abordagem de avaliação. Em vez de comparar o ganho de um indivíduo ou mesmo a média da população, tornou-se necessário estabelecer uma métrica capaz de

verificar com segurança como dois conjuntos de soluções se inter-relacionam segundo critérios de eficiência de Pareto.

Considere, por exemplo, os dois conjuntos de soluções apresentados na Figura 6.1. O volume acinzentado representa os pontos que as soluções factíveis do problema ocupam no espaço de objetivos, enquanto a linha contínua contornando a “face” superior direita do volume representa o espaço ocupado pelas soluções da fronteira Pareto-ótima. Os círculos representam soluções de um conjunto de soluções qualquer, ao passo que os quadrados representam soluções de um outro conjunto de soluções arbitrário. As linhas tracejadas que percorrem as soluções mais próximas da fronteira Pareto-ótima são a interpolação das fronteiras de Pareto definidas pelos dois conjuntos.

Figura 6.1: Dois conjuntos de soluções pertencentes ao mesmo espaço de objetivos, com suas respectivas fronteiras de Pareto destacadas.

O exemplo da Figura 6.1 poderia facilmente encontrar um caso análogo na ECLE. Por exemplo, o conjunto representado pelas circunferências poderia ser resultante de um experimento conduzido pelo algoritmo NSGA-II, ao passo que o conjunto representado pelos quadrados poderia ser resultante de um experimento conduzido pelo algoritmo VEGA, ambos maximizando medidas de sensibilidade e precisão – ainda que seja improvável que um espaço de objetivos determinado por essas medidas tenha o formato indicado na figura. A questão de se determinar qual conjunto de regras é melhor segundo critérios de eficiência de Pareto é, portanto, essencial para que se possa comparar diferentes métodos e configurações. Como o foco da otimização multiobjetivo é a descoberta de um conjunto de soluções eficientes e com alta diversidade, decidiu-se que a avaliação experimental verificaria os dois pontos seguintes.

- O principal fator de comparação é a dominância. Dados dois conjuntos de soluções C_1 e C_2 tal que as soluções de C_1 dominem um grande número de soluções em C_2 e a recíproca não seja verdadeira, então C_1 deve ser considerado superior. Caso contrário, deve haver uma tendência a se declarar um empate entre os conjuntos;

- A distribuição das soluções deve ser levada em consideração, especialmente na fronteira de Pareto. Conforme discutido na Seção 3.3, como não é possível efetuar uma decisão entre diferentes soluções não-dominadas sem acrescentar critérios não considerados no processo de otimização, é importante que usuário tenha acesso a um conjunto de soluções tal que se verifica não apenas alta eficiência de Pareto, mas também um número razoável de soluções na fronteira de Pareto para facilitar a tomada de decisão final.

Tendo-se definido esses requisitos, foram consultadas na literatura diferentes medidas para comparar conjuntos de soluções num espaço de objetivos (Zitzler et al., 2000; Esbensen e Kuh, 1996). Foram deslocadas para o segundo plano aquelas que exigiam a análise de grande quantidade de gráficos ou de estatísticas, na esperança de que os métodos de análise selecionados permitissem avaliação experimental mais simples do que vinha sendo realizada até então, restando assim três medidas de dominância que foram investigadas para que se pudesse definir uma delas como métrica de escolha. Quanto às medidas de diversidade, foi inicialmente verificada a utilidade de uma aplicação do *crowding distance*. Em seguida, variações de métricas envolvendo conceitos de distância euclidiana foram elaborados e investigados.

6.2.1 Medidas de Dominância

Com relação à dominância, foram verificadas as medidas cobertura da fronteira de Pareto, porção dominada e a medida C' . Para todos os casos, considere que se deseja comparar N conjuntos de soluções, $C_1, C_2, \dots, C_N$ e que existe um conjunto composto $C_c = C_1 \cup C_2 \cup \dots \cup C_N$ que corresponde à união de todos os conjuntos em análise e que o conjunto P_c contém todas as soluções que se encontram na fronteira de Pareto do conjunto composto.

Presença na fronteira de Pareto: a presença na fronteira de Pareto de um conjunto C_i é a porcentagem das soluções desse conjunto que se encontram na fronteira de Pareto do conjunto composto.

$$P(C_i, P_c) = \frac{|\{s_x \in C_i \text{ tal que } s_x \in P_c\}|}{|C_i|}$$

Quanto maior a presença na fronteira de Pareto de um conjunto, mais indivíduos dos conjuntos restantes esse conjunto domina. Note que presença na fronteira de Pareto não é uma relação simétrica. O fato de um conjunto possuir alta presença na fronteira de Pareto não significa necessariamente que os outros conjuntos terão baixa presença. No caso mais extremo, todas as soluções podem ser não-dominadas no conjunto composto, de forma que a presença na fronteira de Pareto será 1 para todos os conjuntos.

Porção dominada: a porção dominada é uma medida de eficiência de Pareto que avalia o conceito inverso da presença na fronteira de Pareto. Em vez de verificar a porcentagem de soluções não-dominadas, essa medida verifica a porção de soluções dominadas.

$$D(C_i, P_c) = \frac{|\{s_x \in C_i \text{ tal que } s_x \notin P_c\}|}{|C_i|}$$

Quanto maior o valor da porção dominada, mais as soluções do conjunto são dominadas por conjuntos restantes e menos eficiente são as soluções desse conjunto. Por esse motivo, no lugar da porção dominada pode-se utilizar a complementar da porção dominada:

$$\bar{D}(C_i, P_c) = 1 - D(C_i, P_c)$$

Medida C' : a medida C foi proposta por (Zitzler et al., 2000) para avaliar dois conjuntos de soluções. Quando mais de um conjunto é analisado sob a perspectiva da medida C , os conjuntos são comparados par-a-par. Neste trabalho, utiliza-se o complemento da medida C e a medida é levemente modificada para permitir análise de dominância de um conjunto de soluções contra vários outros conjuntos. Por este motivo, diz-se que a medida utilizada é na verdade a medida C' .

Para calcular C' de C_i é interessante definir primeiramente o conjunto complementar de C_i , composto por todas as soluções consideradas, exceto aquelas contidas em C_i :

$$\mathbb{C}(C_i) = C_1 \cup C_2 \cup \dots \cup C_{i-1} \cup C_{i+1} \cup C_{i+2} \dots C_N$$

Então define-se C' :

$$C'(C_i, \mathbb{C}(C_i)) = 1 - \frac{|\{s_x \in C_i \text{ tal que } \exists s_y \in \mathbb{C}(C_i) \text{ que domina } s_x\}|}{|C_i|}$$

Para verificar qual medida de dominância atenderia melhor aos critérios definidos, a ECLE foi executada diversas vezes no conjunto de dados `breast`. Em um primeiro momento, foram feitas quatro execuções com o mesmo conjunto de regras iniciais – mesma população inicial –, mesmo conjunto de medidas – suporte e precisão –, operadores e critério de parada. Os indivíduos de cada população final desse grupo de experimentos foram registrados em um gráfico 2×2 representando o espaço de objetivos, como o mostrado na Figura 6.2.

Observando-se o gráfico, é fácil perceber que a melhor fronteira de Pareto produzida é aquela do conjunto `breast4`, pois a maior parte de sua extensão “cobre” as fronteiras restantes. Buscou-se então verificar qual medida de dominância corresponderia a

Figura 6.2: Regras representadas no espaço de objetivos. Cada ponto no gráfico é associado a algum indivíduo de uma população final construída pela ECLE. Pontos com mesmo símbolo e cor são associados a indivíduos de uma mesma população. Os pontos associados a indivíduos que ocupam a mesma fronteira quando a população é ordenada por não-dominância são conectados por retas. Todos os conjuntos foram construídos com configurações idênticas e todos os conjuntos possuem 30 indivíduos, embora não seja possível distinguir todos os pontos que os representam pois existe sobreposição.

essa conclusão. Os valores de presença na fronteira de Pareto, porção dominada e medida C' foram calculados e, para o conjunto representado na Figura 6.2, são como se verifica Tabela 6.1.

Tabela 6.1: Medidas para os conjuntos de soluções apresentados na Figura 6.2.

Medida	breast1	breast2	breast3	breast4
Presença na fronteira	0.0189	0.0377	0.0189	0.0755
Porção dominada	0.8491	0.7264	0.7736	0.8019
Medida C'	0.1111	0.1212	0.0769	0.4483

Tanto a medida de presença na fronteira de Pareto quanto a medida C' indicaram o conjunto breast4 como sendo o conjunto com maior eficiência geral. A medida da porção dominada, entretanto, apontou o conjunto breast1 como sendo superior aos outros conjuntos.

Disparidades como essa se verificaram em diversos grupos de experimentos, não apenas com a medida de porção dominada, mas com as outras duas medidas também. Alguns testes foram realizados e esse fenômeno foi explicado por duas causas:

- Embora o interesse seja verificar quais conjuntos apresentam melhores soluções nas fronteiras de Pareto, as medidas de dominância consideraram todos os in-

divíduos das populações no cálculo. Com isso, mesmo que um conjunto C_i qualquer possua soluções não-dominadas que de forma geral são melhores que as soluções não dominadas de um conjunto C_j qualquer, é possível que C_i seja avaliado como de menor qualidade se as soluções dominadas de C_i não forem melhores que as soluções de C_j em geral;

- A existência de soluções com valores idênticos em todas as funções objetivo prejudica a análise, pois o gráfico não permite distinguir facilmente pontos nos quais várias soluções estão contidas. Além disso, soluções com valores idênticos em todas as funções objetivo tem pouco valor para o usuário, pois não oferecem alternativas para o usuário.

Com base nessas constatações, foram realizados os seguintes testes sobre o mesmo grupo de experimentos:

- Todos os indivíduos dominados foram removidos antes que fosse realizada a composição dos gráficos e calculados os valores das medidas de dominância;
- De cada grupo de indivíduos com mesmo genótipo – mesmos valores de variáveis de decisão – foram removidos todos, exceto um. Assim, toda a população ficou constituída por indivíduos com genótipos distintos;
- De cada grupo de indivíduos com mesmo fenótipo – mesmos valores de funções objetivo – foram removidos todos, exceto um. Assim, toda a população ficou constituída por indivíduos que representam soluções em pontos distintos do espaço de objetivos.

Foram feitas várias combinações dos testes. Isto é, em uma avaliação foram apenas eliminados os indivíduos dominados, em outra foram removidos os indivíduos de mesmo genótipo, em outra foram removidos indivíduos dominados e indivíduos com mesmo fenótipo etc. Observe que não existe necessidade de remover indivíduos com mesmo genótipo e mesmo fenótipo, pois os primeiros consistem um subconjunto do segundo.

Os testes mostraram que existe consistência da medida C' com a representação gráfica quando são removidos todos os indivíduos dominados e quando grupos de indivíduos de mesmo fenótipos são reduzidos a um único representante antes da análise de dominância. As medidas de presença na fronteira de Pareto e porção dominada apresentam consistência em alguns casos, porém com menor frequência que a medida C' . Para exemplificar, na Figura 6.3 é mostrado o espaço de objetivos que se obtém quando as medidas são aplicadas aos conjuntos de soluções apresentados na Figura 6.2.

Na Tabela 6.2 são apresentadas as medidas de dominância para os conjuntos da Figura 6.3. Desta vez, acrescenta-se à tabela o número de indivíduos restantes após

Figura 6.3: Soluções únicas não-dominadas.

terem sido removidos os indivíduos dominados e que representam soluções “repetidas”.

Tabela 6.2: Medidas para os conjuntos de soluções apresentados na Figura 6.3.

Medida	breast1	breast2	breast3	breast4
Tamanho da população	5	11	3	9
Presença na fronteira	0.0714	0.1429	0.0714	0.2857
Porção dominada	0.8929	0.7500	0.9643	0.9643
Medida C'	0.4000	0.3636	0.6667	0.8889

Como pode-se observar, em todas as medidas verifica-se consistência com o gráfico. Entretanto, na medida da porção dominada, é incorretamente acusado um empate entre os conjuntos breast3 e breast4. Isso porque a medida considera apenas as soluções que se encontram dominadas pelas soluções dos outros conjuntos. Como o conjunto breast3 resultou poucas soluções após a filtragem, a quantidade de indivíduos dominados, proporcionalmente, é pequena.

A medida que apresentou consistência no maior número de casos foi a medida C' . Por esse motivo, optou por utilizar a medida C' nos experimentos deste trabalho, os quais serão relatados na Seção 6.4.

É importante esclarecer que a medida de estimativa da dominância só tem como finalidade verificar qual conjunto de soluções é superior aos demais em termos de eficiência de Pareto. Os valores não são comparáveis dimensionalmente e não permitem, por exemplo, verificar qual é o segundo ou terceiro melhor conjunto em termos de eficiência de Pareto. Para que essa medida seja obtida, é necessário aplicar um processo iterativo de *rankeamento* que consiste em aplicar a medida C' a todos os

conjuntos, selecionar o conjunto – ou conjuntos – superior aos demais, remover todas as soluções pertencentes a esse(s) conjunto(s) e repetir o processo. Com isso, pode-se obter-se um *ranking* que permite comparar todos os conjuntos de soluções com relação a eficiência de Pareto.

6.2.2 Medidas de Diversidade

Com relação à diversidade, foram verificadas as medidas *crowding distance*, distância entre pares, distância entre pares corrigida e distância média do indivíduo. A diversidade é calculada para um conjunto de soluções $C = \{s_1, s_2, \dots, s_L\}$ sem nenhuma dependência de outros conjuntos em análise. A diversidade é calculada considerando-se as soluções como pontos no espaço M -dimensional determinado pelas funções objetivo $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_M(\mathbf{x})$.

As medidas de diversidade utilizam a distância euclidiana no espaço M -dimensional para avaliar a distância entre duas soluções quaisquer s_x e s_y . Para facilitar a compreensão, define-se a função distância $d(x, y)$ a seguir:

$$d(x, y) = \sqrt{\sum_{i=1}^M (f_i(s_x) - f_i(s_y))^2}$$

Considere que as medidas são aplicadas na população “filtrada”, como explicado na Seção 6.2.1. Desse modo, não há soluções dominadas no conjunto e duas soluções não ocupam o mesmo ponto no espaço de objetivos.

Crowding distance: o *crowding distance* é a medida de diversidade utilizada neste trabalho para implementar a segunda etapa de seleção do algoritmo NSGA-II. Calculando-se o *crowding distance* de todas as soluções e calculando-se a média, pode-se estimar um valor de diversidade aproximado. Quanto maior o valor médio do *crowding distance* para um conjunto de soluções C_i , maior a diversidade.

Distância entre pares: para cada par único de soluções (x, y) , calcula-se a distância $d(x, y)$. A média de todas as medidas de distância entre pares é utilizada como estimador de diversidade.

$$D(C) = \frac{2 \times \sum_{i=1}^L \sum_{j=i+1}^L d(i, j)}{L}$$

O princípio por trás desta medida é que, se o conjunto possuir distância média entre todos os pares elevadas, então a fronteira de Pareto é extensa. De fato, pode-se facilmente verificar que, se dois conjuntos possuem o mesmo número de soluções, então aquela na qual se verifica maior distância média entre pares

é a mais extensa. Infelizmente, quando os conjuntos possuem número distinto de soluções, a comparação não é igualmente confiável, como pode ser verificado na Figura 6.4 e na Tabela 6.3.

Distância entre pares corrigida: a distância entre pares corrigida foi concebida para contornar o problema que ocorre com a distância entre pares quando o número de soluções dos conjuntos difere. Esta medida considera a quantidade de soluções do conjunto e a distância média entre as soluções para estimar a densidade das soluções se a fronteira de Pareto fosse uma reta no espaço de objetivos.

$$D'(C) = \frac{L}{D(C)}$$

Observe que esta medida utiliza a distância média como divisor, o que significa que seu valor é indefinido se $D(C) = 0$. Embora $D(C) \xrightarrow{L \rightarrow 0} \infty$, um conjunto contendo uma única solução não apresenta nenhuma solução alternativa ao usuário e, portanto, esse caso deve ser considerado uma singularidade. Portanto, se $L = 0$, considera-se que a distância entre pares corrigida é nula e o conjunto apresenta diversidade nula.

Distância total média: a distância total média é uma pequena variação da distância média, mas que produz resultados muito mais condizentes com a análise gráfica do que as medidas anteriores. Em vez de se calcular a distância média entre todos os pares do conjunto de solução, calcula-se a somatória da distância de cada indivíduo para todos os restantes e obtém-se o valor médio dessas somatórias.

$$D_2(C) = \frac{\sum_{i=1}^L \sum_{j=1, j \neq i}^L d(i, j)}{L}$$

Observe apenas que a análise de consistência é empírica. Ainda se faz necessária uma avaliação da medida para explicar a razão pela qual ela é mais consistente que a outras medidas.

As medidas de diversidade foram submetidas ao mesmo processo de validação discutido na Seção 6.2.1. Para exemplificar, na Figura 6.4 é apresentado o espaço de objetivos para um grupo de experimentos utilizando o conjunto de dados *breast* e seis diferentes métodos de otimização.

Para os conjuntos ilustrados na Figura 6.4, os valores de diversidade na fronteira de Pareto são os valores mostrados na Tabela 6.3.

Diferentemente do que ocorre com a análise gráfica da dominância, não é tão fácil decidir qual curva representa um conjunto com maior diversidade. Entretanto, é fácil observar que o conjunto de soluções *reciprocal*, embora delineie uma curva bastante prolongada no espaço de objetivos, possui pouquíssima diversidade, se comparada com os demais, pois oferece apenas duas soluções alternativas. Entretanto,

Tabela 6.3: Medidas de diversidade para os conjuntos apresentados na Figura 6.4.

Medida	condorcet	mean	nsga2	reciprocal	vega	vegadiv
Tamanho da população	14	16	18	7	13	11
<i>Crowding distance</i>	0.2857	0.2500	0.2044	0.5714	0.2659	0.3508
Distância entre pares	0.0991	0.0933	0.1329	0.3578	0.0628	0.0600
Distância corrigida	141.23	171.45	135.42	19.56	207.13	183.38
Distância total média	1.2887	1.3999	2.2596	2.1470	0.7532	0.5990

observe como as medidas de *crowding factor* e distância entre pares é afetada pela configuração extrema da curva. Esse comportamento foi observado em muitos grupos de experimentos e, por esse motivo, ambas as medidas foram descartadas como métodos de avaliar a diversidade média da fronteira de Pareto.

Para a medida de distância entre pares corrigida, entretanto, a curva que representaria um conjunto com maior diversidade seria a curva descrita pelo conjunto vega, seguida da curva vegadiv e da curva mean. Embora essas curvas sejam de fato mais diversas do que a curva reciprocal e embora elas possuam uma distribuição razoável de indivíduos ao longo de sua extensão, elas não se estendem no espaço de objetivos tanto quanto as curvas nsga2 e condorcet, por exemplo. Assim, embora esta correção permita contornar o problema em se ter uma curva extensa, porém com poucos in-

Figura 6.4: Regras representadas no espaço de objetivos. Todos os conjuntos são referentes a experimentos da ECLE sobre o conjunto de dados breast. Os conjuntos de soluções condorcet, mean e reciprocal foram construídos com composição de *rankings* – respectivamente, pelo método do *ranking* de Condorcet, média aritmética e *ranking* recíproco. O conjunto nsga2 foi construído pelo algoritmo NSGA-II – Seção 5.4.3 – e os conjuntos vega e vegadiv foram construídos com o algoritmo VEGA, sem utilizar e utilizando mecanismos de manutenção de diversidade, respectivamente.

divíduos contidos nela, a medida da distância entre pares corrigida privilegia curvas densas, ainda que estas sejam menos extensas.

A medida de distância total média, por outro lado, foi capaz de captar exatamente a noção de diversidade esperada. Curvas mais extensas e com maior número de indivíduos nela contidos são avaliadas com maior valor de diversidade. Por esse motivo, a medida de distância total média foi utilizada como estimador de diversidade da fronteira de Pareto nos experimentos realizados neste trabalho.

6.3 Configuração dos Experimentos

Para avaliar os métodos de otimização por *rankings* e por eficiência de Pareto foram realizados diversos experimentos em 13 conjuntos de dados. Todos os conjuntos de dados foram selecionados do repositório *online* da Universidade da Califórnia (Blake et al., 1998). As características dos conjuntos são listadas na Tabela 6.4.

Tabela 6.4: Informações sobre os conjuntos de dados usados. A última coluna indica o número de classes de cada conjunto de dados e o número de exemplos em cada classe – os nomes das classes são omitidos.

Conjunto	# Exemplos	# Atributos (quali.quant.)	# Classes (#exemplos)
breast	285	9 (4, 5)	2 (201, 84)
breastw	683	9 (0, 9)	2 (239, 444)
bupa	345	6 (0, 6)	2 (145, 200)
chess	3196	36 (36, 0)	2 (1669, 1527)
ecoli	336	7 (0, 7)	2 (301, 35)
german	1000	20 (13, 7)	2 (300, 700)
glass	214	9 (0, 9)	2 (197, 17)
haberman	289	3 (0, 3)	2 (225, 81)
heart	270	13 (8, 5)	2 (150, 120)
new-thyroid	215	5 (0, 5)	2 (180, 35)
pima	768	8 (0, 8)	2 (267, 501)
sonar	208	60 (0, 60)	2 (111, 97)
tokyo	959	42 (0, 42)	2 (613, 346)

Foram comparados sete métodos de avaliação multiobjetivo: os cinco métodos de composição de *rankings*, o método VEGA e o NSGA-II. O principal objetivo da otimização foi verificar qual método permitia encontrar fronteira de Pareto mais eficiente. Como segundo objetivo, verificou-se também a diversidade média das regras construídas com cada método de otimização. Para o algoritmo NSGA-II, utilizou-se o *crowding distance*, o qual é tipicamente utilizado na fase de seleção intermediária por *crowded tournament selection*. Para os métodos de composição de *rankings* utilizou-se o mesmo mecanismo baseado em incluir uma nova função objetivo apresentado na Seção 6.2.2. O método VEGA foi avaliado em um grupo de experimentos sem nenhum mecanismo de manutenção de diversidade e em outro foi utilizado o mesmo

mecanismo aplicado nos métodos de composição de *rankings*. Para distinguir os dois casos, o primeiro foi identificado como *vega* e o segundo como *vegadiv*.

Em todos os casos, foi utilizada uma composição dos operadores de *crossover* estrutural, local e de atributo. Para todos os operadores a taxa de *crossover* utilizada foi de 60% com $\alpha = 0.3$ para o caso específico do operador de *crossover* local. Foram utilizados os operadores de mutação local e mutação estrutural, ambos com taxa de mutação de 10%. Tanto a taxa de *crossover* quanto a taxa de mutação foram definidas com base em experiências anteriores do autor com a ECLE¹. Finalmente, para os métodos de composição de *rankings*, o operador de seleção utilizado foi o operador de torneio, com tamanho de torneio 4.

Para avaliar os métodos em um grande número de casos, além de utilizar 11 conjuntos de dados, foram definidos cinco cenários de experimentação. Em cada cenário, um conjunto distinto de medidas de qualidade de regras foi utilizado. No primeiro cenário foram utilizadas apenas medidas de qualidade “clássicas” (precisão e suporte). Nos cenários seguintes foram aplicadas medidas de grau de interesse. As medidas utilizadas em cada cenário são apresentadas na Tabela 6.5.

Tabela 6.5: Medidas utilizadas nos diferentes cenários.

Cenário	Medidas
1	Precisão e suporte
2	Novidade, suporte e Laplace
3	Piatetsky Shapiro’s, suporte e precisão
4	Piatetsky Shapiro’s, suporte e Laplace
5	Coefficiente Φ , suporte e precisão

Finalmente, para fins de suporte estatístico, os testes foram realizados com particionamento 5×2 -*Cross-Fold Validation*. Nessa modalidade de particionamento, o conjunto de dados é particionado em dois conjuntos disjuntos, sendo que em um primeiro momento um deles é utilizado para treinamento e o outro utilizado para teste do modelo induzido; em seguida, o treinamento é realizado com o conjunto anteriormente utilizado para teste e vice-versa. Cada um desses pares é chamado *iteração*. Esse particionamento é realizado cinco vezes e valores de média e desvio padrão podem ser calculados para estimar o desempenho do indutor em situações do mundo real. Além disso, para os experimentos realizados na validação deste trabalho, para cada iteração do particionamento a ECLE foi executada dez vezes. Com isso, cada particionamento completo 5×2 resultou em 100 execuções da ECLE.

Cada cenário associado a um conjunto de dados consiste um *diretório*. Por exemplo, todos os experimentos realizados no cenário 2 (novidade, suporte e Laplace) com o conjunto de dados “Breast Wisconsin” (breastw) são agrupados no diretório breastw 2. É importante esclarecer que em todas as execuções experimentais de um mesmo con-

¹Em projeto de iniciação científica executado em 2007 o autor estudou o impacto de parâmetros como taxa de mutação, taxa de *crossover*, α -blend – *crossover* local – etc no desempenho do algoritmo evolutivo

junto de dados foi feito um mesmo particionamento. Assim, cada execução feita em uma iteração de um diretório qualquer utilizando um método de otimização qualquer pode ser pareada a outras seis execuções na mesma iteração e no mesmo diretório, porém tendo sido utilizados outros métodos de otimização. Com isso, pode-se construir, para cada diretório, 100 grupos de conjuntos de soluções finais nos quais se pode aplicar as medidas de dominância e diversidade discutidas nas Seções 6.2.1 e 6.2.2. Os valores de média e desvio padrão de todos esses grupos podem ser utilizados para comparar o desempenho dos métodos de otimização.

6.4 Resultados

Com base nas execuções experimentais, pode-se comparar os métodos de otimização para diferentes conjuntos de dados e cenários de medidas de qualidade e interesse de regras. Na Tabela 6.6 são apresentados os valores médios de tamanhos da população final (size), *rank* dominância pelo critério C' (dom) e diversidade estimadas pela medida de distância total média (div).

Tabela 6.6: Médias de medidas de dominância, diversidade e tamanho de população para todos os conjuntos e cenários de experimentos.

	condorcet	harmonic	mean	median	nsga2	reciprocal	vega	vegadiv
breast	size: 2.7460	size: 2.0700	size: 4.8340	size: 4.4360	size: 7.1320	size: 2.3980	size: 3.9700	size: 4.5420
	dom: 5.9020	dom: 6.2030	dom: 3.3880	dom: 4.0790	dom: 1.9110	dom: 6.3000	dom: 4.2350	dom: 3.9820
	div: 0.3242	div: 0.1639	div: 0.8893	div: 0.6878	div: 1.0342	div: 0.2883	div: 0.4411	div: 0.6464
breastw	size: 2.1720	size: 1.5120	size: 3.3340	size: 2.9640	size: 3.7180	size: 1.8760	size: 2.5740	size: 3.0200
	dom: 4.5950	dom: 5.6500	dom: 3.7480	dom: 4.5760	dom: 2.0090	dom: 5.5300	dom: 5.2270	dom: 4.6650
	div: 0.1031	div: 0.0270	div: 0.3786	div: 0.2276	div: 0.2188	div: 0.0965	div: 0.1744	div: 0.3144
bupa	size: 3.1800	size: 1.7840	size: 5.8000	size: 5.4480	size: 6.5220	size: 2.9420	size: 3.2840	size: 5.1700
	dom: 5.4030	dom: 6.5840	dom: 3.3500	dom: 3.8190	dom: 2.1800	dom: 5.5020	dom: 5.1810	dom: 3.9810
	div: 0.3573	div: 0.0812	div: 0.9475	div: 0.7678	div: 0.7250	div: 0.3879	div: 0.2133	div: 0.4386
chess	size: 3.4120	size: 2.2940	size: 6.1100	size: 5.3760	size: 17.1460	size: 2.3840	size: 6.4980	size: 7.4300
	dom: 5.3950	dom: 6.4160	dom: 3.9290	dom: 4.2310	dom: 1.7680	dom: 6.7410	dom: 4.0770	dom: 3.4430
	div: 0.4796	div: 0.2196	div: 1.4412	div: 1.2019	div: 3.4108	div: 0.3777	div: 0.8512	div: 1.2943
ecoli	size: 1.6440	size: 1.4160	size: 1.9880	size: 2.1960	size: 2.7760	size: 1.5260	size: 1.8540	size: 2.4860
	dom: 5.0360	dom: 5.0480	dom: 4.4010	dom: 4.3350	dom: 3.6940	dom: 5.1980	dom: 4.3930	dom: 3.8950
	div: 0.0966	div: 0.0520	div: 0.1727	div: 0.2088	div: 0.2545	div: 0.0913	div: 0.0985	div: 0.2032
glass	size: 1.2480	size: 1.1320	size: 1.4380	size: 1.4100	size: 2.0880	size: 1.2360	size: 1.4300	size: 1.7920
	dom: 4.7320	dom: 4.9870	dom: 4.1580	dom: 4.7420	dom: 3.5830	dom: 4.8670	dom: 4.5090	dom: 4.4220
	div: 0.0864	div: 0.0258	div: 0.1330	div: 0.1357	div: 0.3116	div: 0.0900	div: 0.0916	div: 0.1764
haberman	size: 2.9200	size: 1.9920	size: 4.5220	size: 4.5300	size: 6.6300	size: 2.7840	size: 4.0480	size: 5.0940
	dom: 5.5860	dom: 6.2350	dom: 4.1730	dom: 4.1620	dom: 2.0520	dom: 5.9770	dom: 4.1880	dom: 3.6270
	div: 0.3229	div: 0.1492	div: 0.7632	div: 0.6342	div: 0.7730	div: 0.3957	div: 0.3761	div: 0.5385
heart	size: 3.0540	size: 2.0840	size: 4.9180	size: 4.4980	size: 6.1280	size: 2.5900	size: 4.1260	size: 4.9320
	dom: 5.3560	dom: 6.7720	dom: 3.6470	dom: 3.8780	dom: 2.1330	dom: 6.3490	dom: 4.2950	dom: 3.5700
	div: 0.4144	div: 0.1715	div: 1.0862	div: 0.8894	div: 1.1794	div: 0.3632	div: 0.6374	div: 0.9242
new-thyroid	size: 1.3900	size: 1.2220	size: 1.2720	size: 1.3800	size: 1.6800	size: 1.3620	size: 1.4820	size: 1.6380
	dom: 4.4580	dom: 4.3780	dom: 5.1190	dom: 4.6720	dom: 3.7360	dom: 4.6390	dom: 4.3410	dom: 4.6570
	div: 0.0415	div: 0.0243	div: 0.0216	div: 0.0425	div: 0.0760	div: 0.0531	div: 0.0589	div: 0.0674
pima	size: 4.8440	size: 2.2060	size: 9.4340	size: 8.8340	size: 17.8680	size: 4.2380	size: 5.5140	size: 8.9320
	dom: 5.5280	dom: 7.1640	dom: 3.5080	dom: 3.5410	dom: 1.2500	dom: 6.0580	dom: 4.8760	dom: 4.0750
	div: 0.6155	div: 0.1373	div: 1.8995	div: 1.6114	div: 3.0004	div: 0.7434	div: 0.4366	div: 1.2101
sonar	size: 1.8820	size: 1.4260	size: 3.0180	size: 2.2080	size: 3.1260	size: 1.7580	size: 1.7820	size: 2.8860
	dom: 5.0690	dom: 5.4700	dom: 3.7680	dom: 4.8970	dom: 3.3690	dom: 5.1640	dom: 4.6630	dom: 3.6000
	div: 0.0950	div: 0.0168	div: 0.2593	div: 0.1245	div: 0.2395	div: 0.0850	div: 0.0408	div: 0.1741
tokyo	size: 1.9880	size: 1.3920	size: 3.4560	size: 2.7040	size: 5.1000	size: 1.6100	size: 1.8740	size: 3.6000
	dom: 4.8550	dom: 5.5910	dom: 3.5720	dom: 4.4090	dom: 2.8280	dom: 5.4960	dom: 5.1980	dom: 4.0510
	div: 0.1155	div: 0.0209	div: 0.5298	div: 0.3246	div: 0.4446	div: 0.0833	div: 0.0888	div: 0.4542

Na Tabela 6.7 são fornecidos os valores médios de desempenho para cada método considerando-se todos os conjuntos de dados e cenários de execução. São fornecidos também os valores de desvio padrão, os quais foram omitidos na Tabela 6.6 por questão de espaço².

²Os resultados tabelados completos poderão ser encontrados em <http://sites.labc.icmc.usp.br/extradata/giusti/diss>.

Tabela 6.7: Médias de todas as execuções experimentais agrupadas por métodos.

Métodos	Médias	Desv.	Métodos	Médias	Desv.
condorcet	size: 2.5400 / 1.1937		nsga2	size: 6.6595 / 2.3363	
	dom: 5.1596 / 1.8337			dom: 2.5428 / 1.5814	
	div: 0.2543 / 0.0239			div: 0.9723 / 0.0509	
harmonic	size: 1.7108 / 0.7521		reciprocal	size: 2.2253 / 1.1297	
	dom: 5.8748 / 1.6416			dom: 5.6517 / 1.7544	
	div: 0.0908 / 0.0131			div: 0.2546 / 0.0275	
mean	size: 4.1770 / 1.7646		vega	size: 3.2030 / 1.4839	
	dom: 3.8968 / 1.8711			dom: 4.5986 / 1.9331	
	div: 0.7102 / 0.0460			div: 0.2924 / 0.0258	
median	size: 3.8320 / 1.7130		vegadiv	size: 4.2935 / 1.8371	
	dom: 4.2784 / 1.9181			dom: 3.9973 / 1.9853	
	div: 0.5714 / 0.0413			div: 0.5368 / 0.0376	

Com os dados das tabelas, consegue-se observar que o NSGA-II tem desempenho médio superior tanto em termos de dominância quanto em diversidade da fronteira de Pareto, obtendo *rank* médio de 2.54 no critério de dominância e diversidade média 0.97. O método VEGA com mecanismo de diversidade também obtém *rank* médio razoável, 3.99, ao passo o método da composição de *rankings* pela média aritmética obtém o segundo melhor valor médio de diversidade, 0.71.

Entretanto, os valores de desvios padrão são razoavelmente elevados, o que dificulta uma conclusão segura quanto à eficiência dos métodos. Por isso, foram realizados testes de hipótese para verificar se há diferença estatisticamente significativa entre os métodos estudados.

Foi realizado o teste estatístico de Friedman³ com a hipótese nula de que todos os métodos de otimização são igualmente eficientes em termos de dominância. Com 95% de confiança o teste de Friedman rejeitou a hipótese e um teste de Nemenyi foi realizado em caráter *post-hoc* para verificar quais métodos apresentam diferenças estatísticas. O mesmo procedimento foi realizado para os dados referentes a dominância e para diversidade. Na Tabela 6.8 é apresentado um sumário do teste de Nemenyi para dominância.

Tabela 6.8: Resultados do teste de hipótese para dominância.

	nsga2	mean	vegadiv	median	vega	condorcet	reciprocal	harmonic
nsga2	N/A			▲	▲	▲	▲	▲
mean		N/A					▲	▲
vegadiv			N/A				▲	▲
median	▼			N/A			▲	▲
vega	▼				N/A			
condorcet	▼					N/A		
reciprocal	▼	▼	▼	▼			N/A	
harmonic	▼	▼	▼	▼				N/A

A leitura da tabela deve ser feita da seguinte forma. Para cada linha, uma célula em branco indica que não existe diferença estatisticamente significativa entre os dois métodos considerando os experimentos realizados. Uma célula marcada com ▲ indica

³O teste de Friedman é o equivalente não-paramétrico de avaliações-repetidas do ANOVA. Verifique (Demsar, 2006) para maiores discussões sobre testes estatísticos em AM.

que os resultados experimentais apontam superioridade estatisticamente significativa do método assinalado na linha sobre o método assinalado na coluna. Equivalentemente, uma célula marcada com ▼ indica inferioridade estatisticamente significativa.

Com base no teste, pode-se inferir que o método NSGA-II apresentou superioridade estatisticamente significativa sobre quase todos os outros métodos de otimização, com exceção dos métodos VEGA associado a mecanismo de diversidade e ao método de composição de *rankings* baseado na média aritmética. Os métodos do *ranking* recíproco e de composição de *rankings* pela média harmônica apresentaram inferioridade estatisticamente significativa com relação a quatro outros métodos.

A mesma avaliação foi realizada para comparar a diversidade dos conjuntos de regras construídos pelos diferentes métodos. Um sumário do teste de Nemenyi é mostrado na Tabela 6.9.

Tabela 6.9: Resultados do teste de hipótese para diversidade.

	nsga2	mean	vegadiv	median	vega	condorcet	reciprocal	harmonic
nsga2	N/A				▲	▲	▲	▲
mean		N/A			▲	▲	▲	▲
vegadiv			N/A			▲	▲	▲
median				N/A				▲
vega	▼	▼			N/A			
condorcet	▼	▼	▼			N/A		
reciprocal	▼	▼	▼				N/A	
harmonic	▼	▼	▼	▼				N/A

Como pode ser observado, o método NSGA-II apresenta superioridade estatisticamente significativa com relação a quatro métodos e superioridade estatisticamente significativa a quase todos os métodos baseados em composição de *rankings*, à exceção da composição pela média aritmética e pela mediana. O método de composição pelo *ranking* médio não apresentou diferença estatisticamente significante NSGA-II com relação à diversidade. Novamente, o método de composição de *rankings* baseado na média harmônica apresentou inferioridade estatisticamente significativa com relação a quatro métodos.

Primeiramente, observa-se que o método do *ranking* de Condorcet não apenas não apresentou superioridade estatisticamente significativa em termos de dominância ou diversidade com relação a nenhum método de otimização, mas também se mostrou significativamente inferior ao método NSGA-II. Além disso, quando se observa os valores médios de *rank* de dominância e diversidade para o método do *ranking* de Condorcet, observa-se que ele tem um desempenho pobre.

O conflito do desempenho do método do *ranking* de Condorcet observado neste trabalho com o desempenho observado em (Giusti et al., 2008) deve-se à mudança na forma de avaliação dos resultados. Naquele estudo a eficiência dos métodos foi avaliada pela seleção de uma única regra de cada conjunto de regras construído pela ECLE, ao passo que neste trabalho todas as regras que consistem soluções presentes na fronteira de Pareto foram consideradas na análise.

Além disso, é curioso que não se tenha observado diferença estatisticamente significativa, em termos de diversidade, dos resultados produzidos pelo método VEGA para os resultados produzidos pelo método VEGA com mecanismo de diversidade associado. Esperava-se que a inclusão de mecanismo de manutenção da diversidade permitisse encontrar conjuntos de soluções muito mais diversos. Não está descartada a possibilidade de que um mecanismo mais eficaz de manutenção da diversidade do que o mecanismo escolhido possa ser associado a todos os métodos de otimização. Um estudo futuro deverá ser realizado para verificar essa hipótese.

6.5 Considerações Finais

Neste capítulo foi apresentado o método de avaliação da hipótese introduzida no Capítulo 1 da Dissertação:

“A utilização de métodos específicos de otimização multiobjetivo possibilita a construção de regras de conhecimento que atendam critérios múltiplos de qualidade definidos pelo usuário.”

A avaliação foi realizada em 13 conjuntos de dados. Para cada conjunto de dados, os experimentos foram realizados sob cinco cenários de critérios de qualidade de regras. Cinco métodos de construção de regras baseados em composição de *rankings* e dois métodos de computação evolutiva multiobjetivo foram analisados. Como explicado nas Seções 6.2.1 e 6.2.2, as diferentes execuções dos sete métodos de construção de regras foram comparadas com relação a relação de dominância entre os conjuntos de regras e a diversidade de cada conjunto. Foi utilizada validação cruzada $5 \times 2 \times 10$ -Fold.

Com a aplicação de testes de hipótese, verificou-se que o método de computação evolutiva multiobjetivo NSGA-II supera todos os métodos de composição de *rankings* com significância estatística, à exceção do método de composição pela média aritmética.

Como foi demonstrado em (Pila, 2007) que a abordagem baseada em composição de *rankings* é capaz de construir regras de conhecimento que atendam às propriedades específicas, a demonstração de que os métodos baseados em otimização multiobjetivo superam os métodos baseados em composição de *rankings* comprova a hipótese.

Conclusão

7.1 Resumo do Projeto e Revisão do Trabalho

A principal proposta deste trabalho consiste na investigação de métodos de otimização multiobjetivo, baseados no critérios de eficiência de Pareto – Seção 3.3 –, para a construção de regras de conhecimento com critérios definidos pelo usuário – Seção 2.5. Além disso, fazem parte do plano de trabalho deste projeto a modificação da ECLE para a utilização de composição de *rankings* em contexto de otimização multiobjetivo – Seção 5.4 – e a comparação da eficiência dos métodos de otimização baseados em *rankings* contra a dos métodos específicos de otimização multiobjetivo – Capítulo 6.

Para a execução do trabalho foi utilizada a biblioteca de classes e ambiente de construção de regras de conhecimento *Evolutionary Computing Learning Environment* – ECLE. A ECLE – Capítulo 5 – foi inicialmente desenvolvida em um trabalho conjunto do qual o autor participou ativamente (Pila et al., 2006a,b,c; Giusti et al., 2006), sendo que a partir de então o autor é seu mantenedor. A ECLE permite que o usuário defina critérios de qualidade – *e.g.* cobertura, precisão e medida de Piatetsky-Shapiro – e configure os parâmetros de um algoritmo genético – Capítulo 4 – para construir regras de conhecimento com base nesses critérios. Anteriormente a este projeto, a ECLE tratava o problema de construção de regras com base em diversos critérios utilizando uma abordagem de conversão de um problema de otimização multiobjetivo em um problema de otimização simples-objetivo. Isso era obtido por meio de técnicas de composição de *rankings* – Seção 3.4. Assim, ao longo da execução deste projeto a ECLE foi modificada para utilizar critérios de eficiência de Pareto e fronteira de Pareto no processo de otimização, constituindo-se assim uma abordagem inteiramente multiobjetivo. Entretanto, os métodos de composição de *rankings* e outros métodos de composição continuam disponíveis na ECLE.

Para cumprir o terceiro objetivo deste projeto, cinco métodos de otimização baseados em composição de *rankings* e dois métodos específicos de computação evolutiva multiobjetivo – VEGA e ECLE – foram avaliados em 13 conjuntos de dados distintos. Os

experimentos foram organizados em cinco cenários distintos, sendo que cada cenário difere dos demais com relação aos critérios – medidas de qualidade – utilizados no processo de construção das regras. Todos os experimentos foram executados com a técnica de validação cruzada $5 \times 2 \times 10$ -Fold. Com um teste de hipótese seguido de um teste post-hoc, os resultados puderam ser comparados considerando-se a relação de dominância entre os conjuntos de regras construídos por cada método e a diversidade das regras construídas.

Como discutido na Seção 6.4, os experimentos permitiram verificar que o método de otimização multiobjetivo NSGA-II foi superior ao VEGA e a todos os métodos baseados em composição de *rankings*, com exceção do método de composição pela média aritmética. Assim verificou-se com sucesso que a utilização de métodos de otimização multiobjetivo pode permitir de construção de regras de conhecimento com base em critérios especificados pelo usuário de forma superior aos métodos baseados em composição de *rankings*.

7.2 Principais Contribuições

Como argumentado no Capítulo 1, algoritmos de classificação são tipicamente utilizados objetivando-se a construção de classificadores genéricos e precisos – o que normalmente se traduz em altas taxas de cobertura e precisão. Porém, embora essa abordagem seja particularmente interessante em situações nas quais o usuário tem interesse em automatizar o processo de tomada de decisão, ela pode deixar a desejar quando o usuário deseja adquirir o conhecimento extraído para si.

Tendo-se em vista esse problema, a principal proposta deste trabalho é a utilização de uma abordagem evolutiva multiobjetivo para construir regras de conhecimento individuais com base em critérios definidos pelo usuário. Teoricamente, definindo-se os critérios “corretos”, pode-se extrair regras de conhecimento muito mais úteis para o estudo do que aquelas extraídas por algoritmos de classificação “clássicos”¹. A segunda proposta deste trabalho é a extensão da biblioteca de classes ECLE para utilizar explicitamente conceitos de otimização multiobjetivo na construção de regras de conhecimento. Uma terceira proposta, a qual na verdade é fundamental para a validação da hipótese deste trabalho, é a comparação de métodos baseados em computação evolutiva multiobjetivo com métodos baseados em composição de *rankings*, no que diz respeito à maximização dos critérios especificados.

A principal contribuição deste trabalho é a verificação, com base em evidências obtidas empiricamente, de que computação evolutiva multiobjetivo é uma abordagem viável e eficiente para construir regras de conhecimento com base em critérios múltiplos definidos pelo usuário. Tal constatação pode ser feita porque foi verificado em (Pila, 2007) que a composição de *rankings* é uma forma viável de se construir regras de

¹Na próxima seção será feita uma discussão sobre essa hipótese.

conhecimento com base em propriedades específicas. Como neste trabalho verificou-se que a utilização de computação evolutiva multiobjetivo pode produzir resultados superiores àqueles obtidos através de composição de *rankings*, pode-se concluir que a constatação é válida.

Outra importante contribuição deste trabalho é a integração de diversos recursos para otimização multiobjetivo à ECLE, bem como a continuidade de sua manutenção e extensão de sua vida útil. A ECLE tem se mostrado bastante promissora na construção de regras de conhecimento com base em critérios múltiplos definidos pelo usuário e sua extensão e manutenção certamente abre espaço para pesquisas futuras. Em breve o código-fonte da ECLE deverá ser publicado sob uma licença permissiva, juntamente com um relatório técnico a ser escrito.

7.3 Limitações e Trabalhos Futuros

A despeito das contribuições deste trabalho serem sólidas, não se pode descartar um certo grau de autocritica no que diz respeito a suas limitações. A principal limitação deste trabalho diz respeito à expectativa de que a utilização de critérios múltiplos especificados pelo usuário permite a extração de conhecimento que pode ser utilizado pelo usuário para adquirir conhecimento do domínio. Tal expectativa parte do princípio que, no pior cenário possível, a utilização de critérios múltiplos especificados pelo usuário representaria ao menos uma vantagem com relação a algoritmos de AM “clássicos”, os quais em geral focam tão-somente cobertura e precisão.

Seria desejável, portanto, um estudo envolvendo especialistas de diferentes domínios e diferentes critérios de qualidade de regra que permitissem verificar se a ECLE é de fato capaz de construir regras que não apenas atendam aos critérios especificados de forma satisfatória, mas que possam auxiliar o especialista a adquirir conhecimento sobre o domínio. Com efeito, durante boa parte da execução deste trabalho, um estudo de caso envolvendo ao menos um especialista figurou entre os objetivos do projeto. Esse estudo foi realizado durante este projeto, tendo sido consultado um especialista no domínio de processamento de linguagem natural em uma base de dados real. Foi solicitado que o especialista fornecesse notas a regras induzidas por alguns sistemas de AM simbólicos segundo sua noção subjetiva de interesse, as quais foram utilizadas para construir um *ranking* das regras induzidas. Essas mesmas regras foram avaliadas segundo as medidas de interesse apresentadas na Seção 2.5.2. Para cada medida foi construído um *ranking* e foi avaliada a correlação entre cada um desses *rankings* e o *ranking* construído pelo especialista. Verificou-se que nenhuma das medidas empregadas foi capaz de capturar a noção de interesse subjetivo do especialista. O autor acredita que tal noção de interesse somente pode ser capturada se ela for explicitamente fornecida como entrada para a ECLE, por exemplo, por meio de impressões gerais (Liu et al., 1997). Entretanto, devido ao que o autor acredita ser justificado pelas características atípicas do domínio explorado – o conjunto

de dados possui cerca de 100 atributos qualitativos, com grande incidência de um mesmo valor –, o estudo com impressões gerais não produziu resultados satisfatórios neste trabalho. Por essa razão, concluiu-se que seria necessário um estudo muito mais aprofundado para que a avaliação de conhecimento interessante pudesse ser abordada de forma científica e empiricamente válida, o que eventualmente levou à não-inclusão desse estudo como parte dos objetivos do trabalho.

Outra limitação diz respeito aos métodos utilizados na avaliação deste projeto. Seria interessante que, além dos métodos NSGA-II e VEGA, outras técnicas de computação evolutiva multiobjetivo fossem estudadas. Entretanto, para os fins deste projeto, considerou-se que a utilização desses dois métodos seria suficientes, visto que o VEGA é o algoritmo evolutivo multiobjetivo mais simples possível e o NSGA-II é frequentemente tido como padrão de comparação.

Assim, ficam definidas como trabalhos futuros a extensão da ECLE para novos métodos de otimização multiobjetivo e a incorporação desses métodos aos experimentos realizados. Além disso, espera-se realizar futuramente um trabalho envolvendo diversos especialistas de diferentes domínios, permitindo explorar o conhecimento de especialistas como “conhecimento de fundo” para o algoritmo evolutivo, possivelmente permitindo a extração de conhecimento que possa ser de fato considerado interessante pelo especialista do domínio.

Referências Bibliográficas

- Agresti, A. (2002). Categorical data analysis. Citado na página 45.
- Alpaydin, E. (2004). *Introduction to Machine Learning*. MIT Press. Citado na página 27.
- Bäck, T., Hammel, U., e Schwefel, H. (1997). Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17. Citado na página 70.
- Baranauskas, J. A. e Batista, G. E. A. P. A. (2000). O Projeto DISCOVER: Idéias Iniciais. Comunicação pessoal. Citado na página 90.
- Batista, G., Milaré, C., Prati, R., e Monard, M. (2006). A comparison of methods for rule subset selection applied to associative classification. *Inteligencia Artificial. Revista Iberoamericana de IA*, (32):29–35. Citado na página 22.
- Batista, G. E. A. P. A. (2001). Sintaxe padrão do arquivo de exemplos do projeto DISCOVER. <http://www.icmc.sc.usp.br/~gbatista/Discover/SintaxePadraoFinal.htm>. Citado na página 90.
- Batista, G. E. A. P. A. (2003). Pré-processamento de Dados em Aprendizado de Máquina Supervisionado. Tese de Doutorado, ICMC-USP. Citado na página 90.
- Batista, G. E. A. P. A. e Monard, M. C. (2003a). An analysis of four missing data treatment methods for supervised learning. *Applied Artificial Intelligence*, 17(5):519–533. <http://www.icmc.usp.br/~gbatista>. Citado na página 32.
- Batista, G. E. A. P. A. e Monard, M. C. (2003b). Descrição da Arquitetura e do Projeto do Ambiente Computacional DISCOVER LEARNING ENVIRONMENT — DLE. Relatório Técnico 187, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_187.pdf. Citado na página 90.
- Batista, G. E. A. P. A., Prati, R. C., Monard, M. C., Giusti, R., e Milaré, C. R. (2007). Classificação associativa utilizando seleção e construção de regras: um estudo comparativo. Em *Encontro Nacional de Inteligência Artificial (ENIA): Anais do Congresso da Sociedade Brasileira de Computação*, páginas 1321–1330. Citado na página 23.

- Bernardini, F. C. (2006). *Combinação de Classificadores Utilizando Medidas de Regras de Conhecimento e Algoritmos Genéticos*. Tese de Doutorado, ICMC/USP, São Carlos, SP. Citado na página [42](#).
- Blake, C., Keogh, E., e Merz, C. (1998). Uci irvine repository of machine learning databases. <http://www.ics.uci.edu/~mlearn/MLRepository.html>. Citado na página [126](#).
- Bloom, B. et al. (1956). Taxonomy of educational objectives, handbook I: Cognitive domain. Citado na página [26](#).
- Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2:353–373. Citado na página [70](#).
- Brin, S., Motwani, R., e Silverstein, C. (1997). Beyond market baskets: Generalizing association rules to correlations. *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, páginas 265–276. Citado na página [46](#).
- Bäck, T., Fogel, D. B., e Michalewicz, T. (2000). *Evolutionary Computation 1 - Basic Algorithms and Operators*. Institute of Physics Publishing - IoP. Citado nas páginas [138](#) and [140](#).
- Carvalho, A. C. P. L. F., Braga, A. P., e Ludermit, T. B. (2003). *Computação Evolutiva*, chapter 9, páginas 225–248. Em [Rezende \(2003\)](#). Citado na página [83](#).
- Clark, P. e Boswell, R. (1991). Rule induction with CN2: Some recent improvements. Em *Proceedings of the Fifth European Conference*, páginas 151–163. Springer Verlag. Citado na página [21](#).
- Coello, C. A. C. (2006). Evolutionary multi-objective optimization: A historical view of the field. *IEEE Comput. Int. Mag.*, páginas 28–36. Citado na página [53](#).
- Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, páginas 37–46. Citado na página [45](#).
- Cronin, H. (1993). *The ant and the peacock: Altruism and sexual selection from Darwin to today*. Cambridge Univ Pr. Citado na página [72](#).
- Dawkins, R. (1976). *The selfish gene*. Oxford University Press, USA. Citado na página [72](#).
- Deb, K. (2000). *Introduction to selection*, chapter 22, páginas 166–171. Volume I of [Bäck et al. \(2000\)](#). Citado na página [79](#).
- Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, New York, NY, USA. Citado nas páginas [53](#), [66](#), [67](#), and [111](#).

- Deb, K., Agrawal, S., Pratap, A., e Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. Em *Parallel Problem Solving from Nature PPSN VI*, páginas 849–858. Springer. Citado na página 110.
- Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30. Citado na página 129.
- Dorigo, M. (1992). Optimization, learning and natural algorithms. *Doctoral dissertation, Politecnico di Milano, Dipartimento di Elettronica, Italy*. Citado na página 70.
- Dorigo, M., Birattari, M., e Stützle, T. (2006). Ant colony optimization: Artificial ants as a computational intelligence technique. Relatório técnico, Universit Libre de Bruxelles. Citado na página 70.
- Esbensen, H. e Kuh, E. (1996). Design space exploration using the genetic algorithm. Em *1996 IEEE International Symposium on Circuits and Systems, 1996. IS-CAS'96., 'Connecting the World'.*, volume 4. Citado na página 118.
- Fayyad, U., Piatetsky-Shapiro, G., e Smyth, P. (1996a). From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54. Citado na página 42.
- Fayyad, U. M., Piatetsky-Shapiro, G., e Smyth, P. (1996b). The kdd process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34. Citado na página 42.
- Freitas, A. (2002). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag. Citado na página 73.
- Freitas, A. A. (1998a). A multi-criteria approach for the evaluation of rule interestingness. Em *Proc.of the International Conference on Data Mining*, páginas 7–20, Rio de Janeiro, RJ. Citado na página 42.
- Freitas, A. A. (1998b). On objective measures of rule surprisingness. Em *Principles of Data Mining & Knowledge Discovery: Proc.of the Second European Symp. Lecture Notes in Artificial Intelligence*, volume 1510, páginas 1–9. Citado nas páginas 42 and 43.
- Gamma, E., Helm, R., Johnson, R., e Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley. Citado na página 90.
- Garcia, A. C. B., Varejão, F. M., e Ferraz, I. N. (2003). *Aquisição de Conhecimento*, chapter 3, páginas 51–88. Volume 1 of [Rezende \(2003\)](#), 1ª edição. Citado na página 28.
- Geng, L. e Hamilton, H. (2006). Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, 38(3):9. Citado na página 43.

- Giusti, R., Batista, G. E. A. P. A., e Prati, R. C. (2008). Evaluating Ranking Composition Methods for Multi-Objective Optimization of Knowledge Rules. Em *Proceedings of the 8th International Conference on Hybrid Intelligent Systems*, páginas 1–6. IEEE Computer Society. A ser publicado. Citado nas páginas 23 and 130.
- Giusti, R., Pila, A. D., e Monard, M. C. (2006). Arquitetura, Projeto e Implementação da Biblioteca Evolutionary Computing Learning Environment (ECLE) para Construir Regras de Conhecimento com Propriedades Específicas. Relatório técnico, Instituto de Ciências Matemáticas e de Computação (ICMC). Citado nas páginas 22, 91, 95, and 133.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company. Citado nas páginas 72 and 111.
- Gomes, A. K. (2002). Qualidade e interessabilidade de conhecimento adquirido utilizando aprendizado simbólico de máquina. Dissertação de Mestrado, ICMC-USP, <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-04072002-144610/>. Citado na página 42.
- Grefenstette, J. (2000). *Proportional selection and sampling algorithms*, chapter 23, páginas 172–180. Volume I of Bäck et al. (2000). Citado na página 79.
- Hagan, M. T., Demuth, H. B., e Beale, M. H. (1996). *Neural Network Design*. PWS Publishing Co., Boston, 1ª edição. Citado na página 30.
- Haupt, R. L. e Haupt, S. E. (1998). *Practical Genetic Algorithms*. Wiley-interscience Publication. Citado nas páginas 72 and 77.
- Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. Prentice Hall. Citado na página 30.
- Herrera, F., Lozano, M., Pérez, E., Sanchez, A. M., e Villar, P. (2002). Multiple crossover per couple with selection of the two best offspring: An experimental study with the blx-alpha crossover operator for real-coded genetic algorithms. Em *IBERAMIA 2002: Proceedings of the 8th Ibero-American Conference on AI*, páginas 392–401, London, UK. Springer-Verlag. Citado na página 101.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press. Second Edition, MIT Press, 1992. Citado na página 72.
- Horst, P. S. (1999). Avaliação do conhecimento adquirido por algoritmos de aprendizado de máquina utilizando exemplos. Dissertação de Mestrado, ICMC-USP. Citado na página 42.
- Kendall, M. G. (1990). *Rank correlation methods*. A Charles Griffin Book. Citado na página 56.
- Lakshminarayan, K., Harp, S. A., e Samad, T. (1999). Imputation of missing data in industrial databases. *Applied Intelligence*, 11:259–275. Citado na página 32.

- Lavrač, N., Flach, P., e Zupan, B. (1999). Rule evaluation measures: A unifying view. Em Springer-Verlag, editor, *Lecture Notes in Artificial Intelligence*, páginas 174–185. Citado nas páginas [25](#), [37](#), [38](#), and [41](#).
- Liu, B., Hsu, W., e Chen, S. (1997). Using general impressions to analyze discovered classification rules. Em *Knowledge Discovery and Data Mining*, páginas 31–36. Citado na página [135](#).
- McGarry, K. (2005). A survey of interestingness measures for knowledge discovery. *The Knowledge Engineering Review*, 20(01):39–61. Citado na página [43](#).
- Michalewicz, Z. (1997). *Genetic Algorithms + Data Structures = Evolution Programs*. IE-Springer-Verlag. Citado nas páginas [71](#), [73](#), and [85](#).
- Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. The MIT Press. Citado nas páginas [69](#) and [71](#).
- Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill, New York. Citado na página [33](#).
- Mitri, S., Floreano, D., e Keller, L. (2009). The Evolution of Information Suppression in Communicating Robots with Conflicting Interests. *PNAS*, 106:15786–15790. Citado na página [76](#).
- Monard, M. C. e Baranauskas, J. A. (2003a). *Conceitos sobre Aprendizado de Máquina*, chapter 4, páginas 89–114. Em [Rezende \(2003\)](#). Citado na página [21](#).
- Monard, M. C. e Baranauskas, J. A. (2003b). *Indução de Regras e Árvores de Decisão*, chapter 5, páginas 115–140. Em [Rezende \(2003\)](#). Citado na página [21](#).
- Mosteller, F. (1968). Association and estimation in contingency tables. *Journal of the American Statistical Association*, 63(321):1–28. Citado na página [45](#).
- Nienhuys-Cheng, S. e De Wolf, R. (1997). *Foundations of inductive logic programming*. Springer Verlag. Citado na página [31](#).
- Piatetsky-Shapiro, G. (1991). Discovery, analysis, and presentation of strong rules. Em Piatetsky-Shapiro, G. e Frawley, W., editors, *Knowledge Discovery in Databases*. AAAI/MIT Press, Cambridge, MA. Citado na página [46](#).
- Pila, A. D. (2007). *Computação Evolutiva para Construção de Regras de Conhecimento com Propriedades Específicas*. Tese de doutorado, ICMC/USP, São Carlos, SP. Citado nas páginas [22](#), [23](#), [79](#), [115](#), [131](#), and [134](#).
- Pila, A. D., Giusti, R., e Monard, M. C. (2006a). Um sistema evolutivo para a construção de regras de conhecimento com propriedades específicas. Em *32a Conferência Latinoamericana de Informatica*, páginas 1–12. Citado nas páginas [91](#) and [133](#).

- Pila, A. D., Giusti, R., e Monard, M. C. (2006b). Uma proposta para criar regras de conhecimento com propriedades específicas. Em Rezende, S. O. e da Silva Filho, A. C. R., editors, *1st Workshop on Computational Intelligence (WCI'06) in the Proceedings of International Joint Conference, 10th Ibero-American Artificial Intelligence Conference, 18th Brazilian Artificial Intelligence Symposium, 9th Brazilian Neural Networks Symposium, IBERAMIA-SBIA-SBRN*, Ribeirão Preto, Brasil. ICMC-USP. Citado na página 133.
- Pila, A. D., Giusti, R., Prati, R. C., e Monard, M. C. (2006c). A multi-objective evolutionary algorithm to build knowledge classification rules with specific properties. Em *HIS*. IEEE Computer Society. Publicado em CD-ROM. Citado nas páginas 23 and 133.
- Prati, R. C., Baranauskas, J. A., e Monard, M. C. (2001). Uma proposta de unificação da linguagem de representação de conceitos de algoritmos de aprendizado de máquina simbólicos. Relatório Técnico 137, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_137.ps.zip. Citado nas páginas 90 and 93.
- Quinlan, J. R. (1988). *C4.5 Programs for Machine Learning*. Morgan Kaufmann Publishers, CA. Citado na página 21.
- Rezende, S. O. (2003). *Sistemas Inteligentes: Fundamentos e Aplicações*. Editora Manole, Barueri, SP, Brasil. Citado nas páginas 138, 139, and 141.
- Rezende, S. O., Pugliesi, J. B., Melanda, E. A., e Paula, M. F. (2003). *Mineração de Dados*, volume 1 of 1, chapter 12, páginas 307–336. Editora Manole, Barueri, SP, Brasil, 1ª edição. ISBN 85-204-1683-7. Citado na página 42.
- Rijsbergen, C. J. V. (1979). *Information Retrieval*. Butterworth-Heinemann Newton, MA, USA. Citado na página 47.
- Russell, S. e Norvig, P. (2003). *Inteligência Artificial*. Campus Editora, São Paulo. Citado nas páginas 27, 29, and 63.
- Schaffer, J. (1984). *Some experiments in machine learning using vector evaluated genetic algorithms*. Tese de doutorado, Vanderbilt University, Nashville, TN, USA. Citado na página 109.
- Silverstein, C., Brin, S., e Motwani, R. (1998). Beyond market baskets: Generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery*, 2(1):39–68. Citado na página 46.
- Tan, P., Kumar, V., e Srivastava, J. (2002). Selecting the right interestingness measure for association patterns. Citado na página 45.
- Tan, P. N. e Kumar, V. (2000). Interestingness measures for association patterns: A perspective. *KDD-2000 Workshop on Post-processing in Machine Learning and Data Mining*. Citado na página 46.

- Turing, A. (1937). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265. Citado na página [26](#).
- Zhang, H. (2004). The optimality of naive Bayes. Em *Proceedings of the Seventeenth Florida Artificial Intelligence Research Society Conference*, páginas 562–567. The AAAI Press. Citado na página [30](#).
- Zheng, Z. e Low, B. T. (1999). Classifying Unseen Cases with Many Missing Values. Em *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, páginas 370–374. Citado na página [32](#).
- Zitzler, E., Deb, K., e Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195. Citado nas páginas [118](#) and [119](#).