Modelagem fuzzy aplicada a um algoritmo de otimização por colônia de formigas para a extração de regras de classificação em dados

Samira Cássia da Cruz Hodnefjell

Juiz de Fora
Setembro de 2011

Modelagem fuzzy aplicada a um algoritmo de otimização por colônia de formigas para a extração de regras de classificação em dados

Samira Cássia da Cruz Hodnefjell

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal de Juiz de Fora.

Orientador: Professor Ilaim Costa

Juiz de Fora
Setembro de 2011

Samira Cássia da Cruz Hodnefjell

Modelagem fuzzy aplicada a um algoritmo de otimização por colônia de formigas para a extração de regras de classificação em dados.

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal de Juiz de Fora.

Orientador: Professor Ilaim Costa

Trabalho de Conclusão de Curso aprovado em 30/09/2011 pela banca composta pelos seguintes membros:

_____
Prof. Ilaim Costa (Instituto de Computação - UFF) – Orientador

_____
Prof. Custódio Gouvêa Lopes da Motta (UFJF)

_____
Prof. Hélio José Corrêa Barbosa (UFJF)

Conceito obtido _____

Juiz de Fora
Setembro de 2011

Agradecimentos

Ao professor Ilaim Costa pelos ensinamentos, por aceitar sem hesitação a prosseguir com a orientação do presente trabalho à distância, até mesmo após seu desligamento da Universidade Federal de Juiz de Fora.

Aos professores Custódio Motta e Hélio Barbosa pelos ensinamentos sobre Data Mining e Metaheurística e por aceitarem compor a banca de avaliação.

Ao meu esposo Audun Hodnefjell, por todo incentivo, motivação e por todas as inúmeras vezes que ficou espontaneamente encarregado de fazer o jantar enquanto eu desenvolvia o presente trabalho.

Aos meus sogros Vigdis e Ole Ingvar Hodnefjell pelo apoio e por limitarem as brincadeiras de meus pequenos sobrinhos, assíduos visitantes nos fins de semana, para que o barulho não me perturbasse.

A toda minha família e amigos residentes no Brasil pelo apoio e compreensão da minha frequente falta de disponibilidade para colocar a conversa em dia.

A todos os amigos, colegas de classe e estudiosos que serviram de inspiração.

Sumário

Lista de Tabelas

Lista de Figuras

Resumo

O algoritmo *Ant-Miner*, originalmente proposto por Parpinelli, Lopes e Freitas (2002a), aplica a metaheurística da otimização por colônia de formigas para a tarefa de classificação em mineração de dados. O presente trabalho aplica a modelagem fuzzy do comportamento forrageiro das formigas, proposto por Rozin e Margaliot (2007), ao algoritmo *Ant-Miner* admitindo duas possíveis interpretações e adicionando a elas elementos como a função heurística, presente no *Ant-Miner* original, e a técnica de priorização ao atributo utilizado, apresentada neste trabalho. Resultantes dessas adaptações, foram geradas 8 diferentes versões do algoritmo *Ant-Miner* que foram testadas e comparadas entre si e à versão original utilizando sete *data sets* de domínio público. Uma das versões apresentou resultados gerais superiores às outras, incluindo o algoritmo *Ant-Miner* original, em termos de acurácia preditiva em diferentes configurações do sistema.

Palavras-chave: mineração de dados, regras de classificação, metaheurística otimização por colônia de formigas, modelagem *fuzzy*.

Abstract

The Ant-Miner algorithm, originally proposed by Parpinelli, Lopes and Freitas (2002a), applies the ant colony optimization metaheuristic for the classification task in data mining. This project applies the fuzzy modeling of the foraging behavior of ants, proposed by Rozin and Margaliot (2007), to the Ant-Miner algorithm admitting two possible interpretations and adding to them elements such as the heuristic function, present in the original Ant-Miner, and the used attribute priority technique, presented in this project. As a result of these adaptations, 8 different versions of the Ant-Miner algorithm were generated, tested and compared to each other and to the original version using seven public domain data sets. One of the versions produced comparatively superior results to the others, including the original Ant-Miner algorithm, in terms of predictive accuracy in different system configurations.

Key-words: data mining, classification rules, ant colony optimization metaheuristic, fuzzy modeling.

Capítulo 1 - Introdução

O presente trabalho relaciona três assuntos distintos que serão brevemente apresentados neste capítulo, sendo eles: a otimização por colônia de formigas, a classificação em mineração de dados e a modelagem *fuzzy*.

O algoritmo *Ant-Miner*, proposto por Parpinelli, Lopes e Freitas (2002a), foi o primeiro algoritmo a utilizar otimização por colônia de formigas (*ACO*, do inglês *ant colony optimization)* para descobrir regras de classificação em mineração de dados.

A otimização por colônia de formigas, segundo Dorigo e Stutzle (2004) é um paradigma da inteligência computacional inspirado no comportamento natural das formigas. Uma colônia de formigas tem a capacidade de encontrar o caminho mais curto entre o formigueiro e uma fonte de alimento sem utilizar informação visual. A comunicação entre as formigas é feita através de uma substância química denominada feromônio. Ao se deslocarem, as formigas depositam uma certa quantidade de feromônio no solo, criando as chamadas trilhas de feromônio. Quanto mais formigas seguirem uma determinada trilha, mais atraente a trilha se torna para que outras formigas passem a seguir a mesma trilha. Este processo envolve um ciclo de *feedback* positivo, em que a probabilidade de uma formiga escolher um caminho é proporcional ao número de formigas que já passou por esse caminho.

Desta forma, cada formiga, individualmente, segue regras muito simples, mas é sua interação com as outras que leva ao surgimento de um comportamento altamente inteligente no nível da colônia.

A tarefa de classificação em mineração de dados consiste em prever o valor do atributo classe de cada instância pertencente a um determinado conjunto de registros de dados, através da análise dos valores dos outros atributos de cada instância.

A modificação aplicada ao *Ant-Miner* no presente trabalho se refere à implementação da modelagem *fuzzy*. A lógica *fuzzy*, também conhecida como lógica difusa ou lógica nebulosa, trata de aspectos vagos da informação, admitindo valores lógicos intermediários a 0 e 1 (falso e verdadeiro) da lógica booleana.

1.1 Motivação

As bases de dados vêm crescendo exponencialmente nos últimos anos resultando em um número exorbitante de dados. Analisar eficientemente essa enorme quantidade de dados na busca por informações relevantes se tornou uma tarefa de extrema importância. A mineração de dados surgiu dessa necessidade e, atualmente, consiste em uma área em constante ascensão. Daí o interesse em explorar meios alternativos de promover a mineração de dados.

A utilização de algoritmos que aplicam a otimização por colônia de formigas (*ACO*) para a descoberta de regras de classificação, assim como outras tarefas relacionadas à mineração de dados, pode apresentar uma vantagem significativa em relação à utilização de algoritmos determinísticos de indução de regra.

Tal informação se justifica pelo fato de que esses últimos algoritmos normalmente seguem a heurística gulosa e, por isso, são suscetíveis a encontrar regras de classificação apenas ótimas localmente. Em contraste, os algoritmos *ACO* combinam duas ideias básicas que visam atenuar essa possibilidade. A primeira é seu aspecto estocástico, que os ajuda a explorar uma área maior do espaço de busca. A segunda é o procedimento iterativo de adaptação com base no *feedback* positivo, ou seja, o aumento gradual do feromônio associado aos componentes da melhor solução, para melhorar continuamente as regras candidatas.

Assim, em geral, os algoritmos *ACO* realizam uma pesquisa mais global no espaço de regras candidatas do que os algoritmos determinísticos típicos de indução de regras, os tornando uma alternativa interessante a ser considerada na descoberta de regras de classificação.

A modelagem *fuzzy* pode ser aplicada de forma simples e eficiente para transformar descrições verbais e explicações de fenômenos naturais em rigorosos modelos matemáticos. O comportamento forrageiro das formigas apresenta uma característica *fuzzy* pela aleatoriedade da forma em que elas se movimentam e, além disso, tal comportamento pode ser convertido em uma descrição verbal. Assim, a modelagem *fuzzy* consiste em uma ferramenta apropriada para estudar o comportamento das formigas.

1.2 Hipótese

Podemos aprimorar o algoritmo *Ant-Miner* original ao alterá-lo implementando a modelagem *fuzzy* e adaptando-a sistematicamente, de diversas formas.

1.3 Objetivos

1.3.1 Objetivo geral

Verificar a possibilidade de aprimoramento do algoritmo *Ant-Miner* original através de determinadas modificações aplicadas ao mesmo, baseadas na modelagem *fuzzy*.

1.3.2 Objetivos específicos

- Apresentar a base teórica sobre o qual o algoritmo *Ant-Miner* foi desenvolvido;
- Propor modificações ao algoritmo baseadas em modelagem *fuzzy,* aplicá-las e analisar os resultados obtidos;
- Investigar outras modificações que, possivelmente, levem à melhoria dos resultados apresentados pelo algoritmo original;
- Analisar os resultados sistematicamente de forma a facilitar o desenvolvimento de trabalhos futuros.

Os capítulos a seguir apresentam detalhadamente os assuntos abordados neste trabalho. O Capítulo 2 introduz brevemente a mineração de dados, focando na tarefa de classificação. O Capítulo 3 aborda a otimização por colônia de formigas. O Capítulo 4 faz uma breve introdução sobre modelagem *fuzzy*. O algoritmo *Ant-Miner* é apresentado no Capítulo 5. As modificações aplicadas são expostas no Capítulo 6. O Capítulo 7 apresenta os resultados e o Capítulo 8 as conclusões e os trabalhos futuros.

Capítulo 2 - Mineração de Dados

Desde que os dados passaram a ser coletados e armazenados em meio magnético, concomitantemente ao avanço da tecnologia que permite armazenar um volume de dados cada vez maior, até os dias atuais, temos observado uma explosão quantitativa das bases de dados. Neste cenário, surge um problema eminente: como analisar eficientemente esse número exorbitante de dados? Naisbitt (1990) ilustra bem esse problema ao concluir que estamos afogados em informação, mas famintos por conhecimento.

Uma das principais soluções que emergiram dessa necessidade é a Mineração de Dados ou *Data Mining*, que compõe o processo de Descoberta de Conhecimento em Bases de Dados ou *Knowledge Discovery in Databases* (*KDD*).

> O processo de descoberta de conhecimento em bases de dados é definido como sendo um processo interativo e iterativo, não trivial de identificação de padrões válidos, novos, potencialmente úteis, compreensíveis e embutidos nos dados, envolvendo numerosos passos, com muitas decisões sendo feitas pelo usuário. (FAYYAD, PIATETSKY-SHAPIRO, SMITH, UTHURUSAMY, 1996, p. 5)

A mineração de dados é um campo interdisciplinar que relaciona principalmente técnicas de aprendizagem de máquina, estatística e bancos de dados. Seu objetivo é, em essência, extrair conhecimento, — ou, mais especificamente, padrões estruturais —, a partir de dados como uma ferramenta para ajudar a explicá-los e fazer previsões através dos mesmos.

Tal conhecimento deve ser preciso e compreensível para o usuário, de forma que seja possível utilizá-lo para a tomada de decisões. Assim, o usuário deve ser capaz de interpretar e validar o conhecimento adquirido.

Existem várias tarefas de mineração de dados, como a classificação, associação, regressão, *clustering*, etc. Cada uma delas representa um tipo de problema específico que demanda um determinado tipo de algoritmo destinado à resolução do problema. Nesses termos, na concepção de um algoritmo de

mineração de dados é importante primeiramente definir a qual tarefa o algoritmo se destina. Neste trabalho, enfocaremos a tarefa de classificação.

2.1 Conceitos Básicos

A Tabela 2.1 exibe um exemplo clássico de base de dados que relaciona as condições climáticas à recomendação do jogo de tênis. Utilizaremos este exemplo para definir alguns conceitos básicos em mineração de dados.

Aspecto	Temperatura	Humidade	Vento	Jogar
ensolarado	quente	alta	falso	não
ensolarado	quente	alta	verdadeiro	não
nublado	quente	alta	falso	sim
chuvoso	moderada	alta	falso	sim
chuvoso	fria	normal	falso	sim
chuvoso	fria	normal	verdadeiro	não
nublado	fria	normal	verdadeiro	sim
ensolarado	moderada	alta	falso	não
ensolarado	fria	normal	falso	sim
chuvoso	moderada	normal	falso	sim
ensolarado	moderada	normal	verdadeiro	sim
nublado	moderada	alta	verdadeiro	sim
nublado	quente	normal	falso	sim
chuvoso	moderada	alta	verdadeiro	não

Tabela 2.1 Os dados do arquivo *Jogar tênis* (WITTEN & FRANK, 2005)

Uma instância é um exemplo individual e independente do conceito a ser aprendido. Na Tabela 2.1, cada linha representa uma instância distinta. Cada linha da tabela pode ser interpretada como um determinado momento em que dadas as condições climáticas, optou-se por jogar ou não tênis.

Um atributo é um valor de uma característica fixa e pré-definida de uma instância. Na Tabela 2.1, cada coluna representa um atributo distinto. Os atributos podem ser do tipo: nominal, também chamados categóricos, como os presentes no exemplo da Tabela 2.1, em que o valor de cada atributo deve ser escolhido de um conjunto limitado de possíveis valores; numéricos, também chamados contínuos; dentre outros. O algoritmo *Ant-Miner* processa somente atributos nominais.

Uma classe é um atributo que deve ser predito. No caso do exemplo da Tabela 2.1, o objetivo mais usual é prever a possibilidade de jogar tênis.

Os dados são geralmente divididos em um conjunto de treinamento e um conjunto de testes, ambos mutuamente independentes. O conjunto de treinamento é usado para criar o modelo de aprendizado e o de teste para testar a acurácia do modelo criado.

2.2 Classificação

Um classificador, primeiramente, gera um modelo constituído de regras de classificação que classifica as instâncias de um conjunto de treinamento em um número pré-determinado de classes. Os possíveis valores que a classe pode assumir assim como o próprio atributo classe devem ser fornecidos. Por essa razão, o processo de aprendizado na classificação é dito supervisionado.

Em seguida, o modelo criado deve ser testado sobre as instâncias do conjunto de teste e sua qualidade deve ser medida calculando a porcentagem de acertos, ou seja, a porcentagem do número de instâncias classificadas de forma satisfatória pelas regras geradas.

Ao se obter uma acurácia desejada, o modelo pode ser aplicado a uma nova amostra de dados que possua classes desconhecidas, da qual se deseja, efetivamente, obter os resultados da classificação.

2.2.1 Representação do conhecimento

Existem dois estilos básicos de representação do conhecimento derivados da classificação que são amplamente usados por métodos de aprendizagem de máquina. São eles as árvores de decisão e as regras de classificação. Suas estruturas serão brevemente apresentadas a seguir.

2.2.1.1 Árvores de decisão

Uma árvore de decisão é naturalmente derivada de uma abordagem de divisão e conquista.

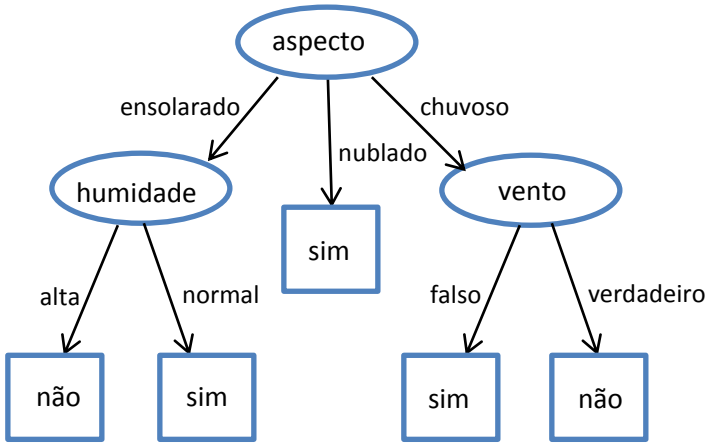

Figura 2.1 Árvore de decisão para os dados Jogar tênis (WITTEN & FRANK, 2005)

Os nós folhas de uma árvore de decisão representam os valores do atributo classe, em outras palavras, representam a classificação aplicada a todas as instâncias que atingem o nó folha. Os demais atributos são representados pelos nós internos. Cada ramo que liga um nó-pai a seus nós-filhos possui um dos possíveis valores que o atributo do nó-pai pode assumir. O nó que representa um determinado atributo não pode ser descendente do mesmo atributo em questão.

2.2.1.2 Regras de classificação

As regras de classificação são condições lógicas que possuem um antecedente e um consequente. O antecedente, como os nós da árvore de decisão, representam os testes realizados sobre os atributos e o consequente determina a classe que se aplica à instância coberta pela regra. Geralmente, o antecedente é composto pelo operador lógico *E,* assim, todos os termos devem ser verdadeiros para que a regra possa ser aplicada.

A Figura 2.2 mostra a árvore de decisão da Figura 2.1 transformada em regras de classificação.

SE aspecto = ensolarado **E** humidade = alta **ENTÃO** jogar = não
SE aspecto = ensolarado **E** humidade = normal **ENTÃO** jogar = sim
SE aspecto = nublado **ENTÃO** jogar = sim
SE aspecto = chuvoso **E** vento = falso **ENTÃO** jogar = sim
SE aspecto = chuvoso **E** vento = verdadeiro **ENTÃO** jogar = não

Figura 2.2 Regras de classificação para os dados Jogar tênis (WITTEN & FRANK, 2005)

2.2.2 Algoritmos de classificação

Alguns dos métodos mais utilizados destinados à classificação em mineração de dados são a modelagem estatística, redes neurais, métodos baseados em divisão e conquista, — que gera árvores de decisão — e o algoritmo de cobertura que será apresentado a seguir.

2.2.2.1 Algoritmo de Cobertura

O algoritmo de cobertura funciona basicamente selecionando cada uma das classes em iterações distintas e procurando uma maneira de cobrir todas as instâncias pertencentes à classe corrente, ao mesmo tempo excluindo as instâncias que não pertencem à classe em questão. Em cada estágio, o algoritmo identifica uma regra que é válida para um grupo de instâncias.

O algoritmo opera adicionando testes à regra em construção na tentativa de criar uma regra com a máxima acurácia. Diferentemente, o algoritmo de divisão e conquista adiciona testes à árvore em construção na tentativa de maximizar a separação entre as classes. Ambos procuram um atributo para divisão, mas selecionam tal atributo através de critérios diferentes. O algoritmo de divisão e conquista escolhe um atributo que maximize o ganho da informação enquanto o algoritmo de cobertura seleciona um par atributo-valor que maximize a probabilidade da classificação desejada.

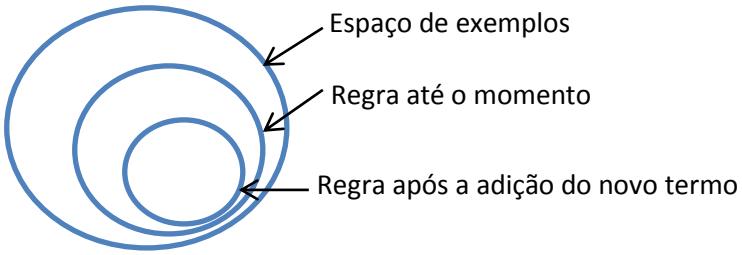

Figura 2.3 O espaço de instâncias durante a operação de cobertura (WITTEN & FRANK, 2005)

A Figura 2.3 mostra o espaço contendo todas as instâncias, uma regra parcialmente construída e a mesma regra após a adição de um novo termo. O objetivo é incluir o maior número de instâncias possível da classe desejada e excluir o maior número possível de instâncias de outras classes, assim, como podemos observar, a adição do novo termo restringe a cobertura da regra. Supondo que a nova regra cubra um total de *t* instâncias, nas quais *p* são exemplos positivos da classe e *t* – *p* são os erros cometidos pela regra. O novo termo deve maximizar a razão *p/t*.

A seguir apresentamos um simples exemplo usando os dados da Tabela 2.1. Trata-se do método de construção de regras denominado PRISM. Tal método continua adicionando termos à regra até que ela atinja acurácia máxima. Iniciaremos com a seguinte regra:

Se *?* ***então*** *Jogar = sim*

Para o termo desconhecido ?, temos dez escolhas:

Aspecto = nublado	4/4
Aspecto = chuvoso	3/5
Aspecto = ensolarado	2/5
Temperatura = quente	2/4
Temperatura = moderada	4/6

Temperatura = fria	3/4
Humidade = alta	3/7
Humidade = normal	6/7
Vento = falso	6/8
Vento = verdadeiro	3/6

As frações à direita representam o número de instâncias corretamente classificadas pelo termo, (pertecem à classe Jogar = sim), dividido pelo número total de instâncias que o termo seleciona. Por exemplo, o termo Aspecto = chuvoso seleciona 5 instâncias e 3 delas pertencem à classe Jogar = sim.

Selecionaremos assim, a maior fração, criando a regra:

Se *Aspecto = nublado* ***então*** *Jogar = sim*

Essa regra tem acurácia de 100%, dessa forma, podemos excluir as instâncias cobertas por ela e reiniciar o processo para a criação da próxima regra. Após a exclusão das 4 instâncias, recalculamos as frações *p/t* obtendo:

Aspecto = chuvoso	3/5
Aspecto = ensolarado	2/5
Temperatura = quente	0/2
Temperatura = moderada	3/5
Temperatura = fria	2/3
Humidade = alta	1/5
Humidade = normal	4/5
Vento = falso	4/6
Vento = verdadeiro	1/4

Mais uma vez, selecionamos a maior fração e obtemos:

***Se** Humidade = normal **então** Jogar = sim*

Aspecto	Temperatura	Humidade	Vento	Jogar
chuvoso	fria	normal	falso	sim
chuvoso	fria	normal	verdadeiro	não
ensolarado	fria	normal	falso	sim
chuvoso	moderada	normal	falso	sim
ensolarado	moderada	normal	verdadeiro	sim

Tabela 2.2 Parte dos dados *Joga tênis* para os quais Humidade = normal

Essa regra seleciona 4 de 5 instâncias, exibidas na Tabela 2.2. Como sua acurácia ainda não é máxima, devemos refiná-la adicionando a ela um novo termo:

***Se** Humidade = normal **e** ? **então** Jogar = sim*

Temos seis possibilidades para o termo ?, obtidas através do cálculo das razões *p/t* considerando apenas as instâncias já selecionadas pela regra, exibidas na Tabela 2.2. Assim:

Aspecto = chuvoso	2/3
Aspecto = ensolarado	2/2
Temperatura = moderada	2/2
Temperatura = fria	2/3
Vento = falso	3/3
Vento = verdadeiro	1/2

Há um empate entre o segundo, terceiro e quinto termos, pois suas frações são iguais a 1, porém selecionaremos o quinto termo pois sua cobertura é maior (3 de 3), obtendo:

***Se** Humidade = normal **e** Vento = falso **então** Jogar = sim*

Novamente, obtemos uma regra com acurácia máxima. As duas regras descobertas até o momento cobrem 7 de 9 instâncias cuja classe é sim. As instâncias que ainda devem ser consideradas para a continuação do processo estão na Tabela 2.3.

Aspecto	Temperatura	Humidade	Vento	Jogar
ensolarado	quente	alta	falso	não
ensolarado	quente	alta	verdadeiro	não
chuvoso	moderada	alta	falso	sim
chuvoso	fria	normal	verdadeiro	não
ensolarado	moderada	alta	falso	não
ensolarado	moderada	normal	verdadeiro	sim
chuvoso	moderada	alta	verdadeiro	não

Tabela 2.3 Parte dos dados *Joga tênis* após a aplicação das duas regras.

Recalculando as razões *p/t* utilizando as instâncias listadas na Tabela 2.3, obtemos:

Aspecto = chuvoso	1/3
Aspecto = ensolarado	1/4
Temperatura = quente	0/2
Temperatura = moderada	2/4
Temperatura = fria	0/1
Humidade = alta	1/5
Humidade = normal	1/2
Vento = falso	1/3
Vento = verdadeiro	1/4

Iniciaremos então mais uma regra selecionando a maior fração e de maior cobertura:

Se *Temperatura = moderada* ***então*** *Jogar = sim*

Repetindo o processo explicado anteriormente, refinaremos a regra através da escolha de umas das possibilidades:

Aspecto = chuvoso	1/2
Aspecto = ensolarado	1/2
Humidade = alta	1/3
Humidade = normal	1/1
Vento = falso	1/2
Vento = verdadeiro	1/2

Assim, obtemos:

Se *Temperatura = moderada* ***e*** *Humidade = normal* ***então*** *Jogar = sim*

Mais uma vez, obtemos uma regra com acurácia máxima, porém ela cobre somente uma instância. É interessante permitir a definição de parâmetros que irão determinar a qualidade desejada das regras extraídas. Neste exemplo estamos apenas interessados em obter regras válidas para 100% dos casos.

Aspecto	Temperatura	Humidade	Vento	Jogar
ensolarado	quente	alta	falso	não
ensolarado	quente	alta	verdadeiro	não
chuvoso	moderada	alta	falso	sim
chuvoso	fria	normal	verdadeiro	não
ensolarado	moderada	alta	falso	não
chuvoso	moderada	alta	verdadeiro	não

Tabela 2.4 Parte dos dados *Joga tênis* após a aplicação das 3 regras

Como podemos observar na Tabela 2.4, após a exclusão da instância coberta ainda existe uma instância da classe Jogar = sim para a qual devemos construir uma regra. Assim, dando continuidade ao processo:

Aspecto = chuvoso	1/3
Aspecto = ensolarado	0/3
Temperatura = quente	0/2
Temperatura = moderada	1/3
Temperatura = fria	0/1
Humidade = alta	1/5
Humidade = normal	0/1
Vento = falso	1/3
Vento = verdadeiro	0/3

Escolheremos arbitrariamente entre o primeiro, quarto e oitavo termos, obtendo a regra que cobrirá a última instância da classe Jogar = sim:

Se *Aspecto = chuvoso* ***então*** *Jogar = sim*

Essa regra tem acurácia de 1/3 e deve ser refinada. Recalculando as razões *p/t*, com base nas 3 instâncias selecionadas, obtemos:

Temperatura = moderada	1/2
Temperatura = fria	0/1
Humidade = alta	1/2
Humidade = normal	0/1
Vento = falso	1/1
Vento = verdadeiro	0/2

Logo, o termo Vento = falso, deve ser adicionado à regra atribuindo a ela acurácia máxima. Desta forma, obtemos 4 regras que classificam todas as instâncias do conjunto de treinamento cuja classe é sim:

Se *Aspecto = nublado* ***então*** *Jogar = sim*
Se *Humidade = normal* ***e*** *Vento = falso* ***então*** *Jogar = sim*
Se *Temperatura = moderada* ***e*** *Humidade = normal* ***então*** *Jogar = sim*
Se *Aspecto = chuvoso* ***e*** *Vento = falso* ***então*** *Jogar = sim*

O próximo passo é repetir, da mesma forma, o processo para a classe Jogar = não.

```
Para cada classe C
    Inicialize E com o conjunto de instâncias
    Enquanto E possuir instâncias na classe C
        Crie uma regra R com antecedente vazio que prediga a classe C
        Até que R seja perfeita (ou não exista mais atributos a serem utilizados) faça
            Para cada atributo A não mencionado em R, e cada valor v,
                Considere adicionar a condição A=v à regra R
                Selecione A e v para maximizar a acurácia p/t (em caso de empate
                escolher o maior p)
            Adicione A=v à R
        Remova as instâncias cobertas por R de E
```

Figura 2.4 Pseudocódigo do algoritmo PRISM (WITTEN & FRANK, 2005)

A Figura 2.4 apresenta em síntese o pseudocódigo do algoritmo. O laço externo gera regras para cada classe. No início de sua execução, o conjunto de exemplos é reestabelecido com o conjunto completo de instâncias. As regras são criadas para a classe corrente e os exemplos cobertos são removidos até que não sobre nenhum pertencente à classe atual. Ao criar a regra, inicia-se com uma regra vazia que é então restringida pela adição de termos até que cubra somente exemplos da classe desejada.

O método PRISM descrito pode ser classificado como um algoritmo de separação e conquista. Tal afirmação se justifica pelo fato do algoritmo identificar uma regra que cubra um certo número de instâncias pertencentes a uma classe e separar as instâncias já cobertas pela regra, continuando o processo com as instâncias restantes.

O algoritmo *Ant-Miner* é, estruturalmente, um algoritmo de cobertura que não possui o laço externo presente no algoritmo da Figura 2.4. Desta forma, ele gera regras que devem ser interpretadas em ordem, ou seja, uma lista de decisão.

Capítulo 3 - Otimização por Colônia de Formigas

Um algoritmo *ACO* é, essencialmente, um sistema baseado em agentes que simulam o comportamento natural de uma colônia de formigas, incluindo principalmente mecanismos de cooperação e de adaptação. A utilização desse tipo de sistema como uma nova metaheurística foi proposta em 1999 por Dorigo e Di Caro para resolver problemas de otimização combinatória. Esta nova metaheurística tem-se mostrado robusta e versátil, tendo sido aplicada com sucesso a uma gama de diferentes problemas de otimização combinatória.

3.1 A Inspiração biológica

As formigas são seres extremamente simples e têm uma capacidade limitada de processar e trocar informação. No entanto, é de suma importância que elas encontrem o menor caminho entre o formigueiro e uma fonte de alimento, para reduzir seus esforços e sua exposição a perigos, e também que encontrem um consenso sobre qual caminho seguir sem utilizar qualquer ordem hierárquica.

De fato, na maioria das vezes, as formigas conseguem encontrar esse caminho e adaptam-se também às mudanças no ambiente sem utilizar informação visual. Tal capacidade tem sido estudada extensivamente por etólogos. Seus estudos revelaram que, a fim de trocar informações sobre qual caminho deve ser seguido, muitas espécies de formigas, assim como outros insetos sociais, se comunicam umas com as outras por meio de trilhas de uma substância química denominada feromônio. Ao se deslocarem entre o ninho e a fonte de alimento, as formigas depositam uma certa quantidade de feromônio no solo, marcando o caminho com uma trilha dessa substância. As outras formigas têm a capacidade de perceber a presença do feromônio e tendem a seguir o caminho cuja concentração

de feromônio é maior. Desta forma, as formigas conseguem transportar o alimento para o formigueiro de maneira bastante efetiva.

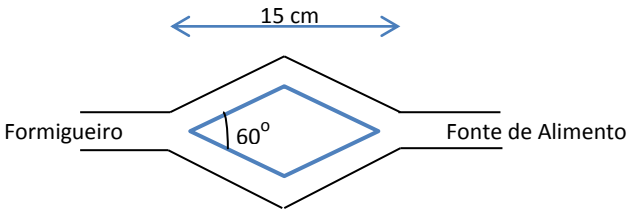

Figura 3.1 Experimento da Ponte Dupla com pontes de mesmo tamanho (DORIGO, BIRATTARI & STÜTZLE, 2006)

O comportamento das formigas foi estudado por Deneubourg et al. (1989) através do experimento da ponte dupla. Nesse experimento, realizado com uma colônia de formigas argentinas, duas pontes de mesmo tamanho foram colocadas entre o formigueiro e a fonte de alimento, conforme a Figura 3.1.

Inicialmente, cada formiga escolhe uma das pontes aleatoriamente. Depois de algum tempo, devido a fatores aleatórios, uma das pontes passará a apresentar uma concentração maior de feromônio, atraindo mais formigas, o que acarretará na atração de cada vez mais formigas, fazendo com que, após algum tempo, a colônia inteira convirja para a utilização da mesma ponte.

Uma variação do experimento anterior foi realizada por Goss et al. (1990), na qual uma das pontes é significantemente maior que a outra (Figura 3.2).

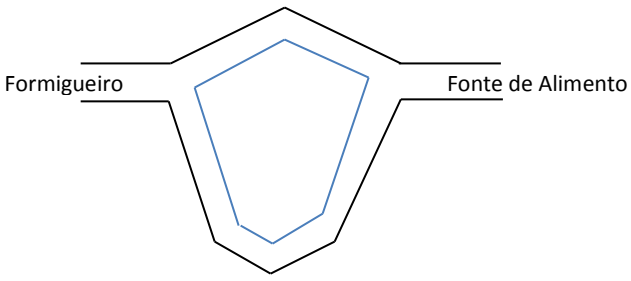

Figura 3.2 Experimento da Ponte Dupla com pontes de tamanhos diferentes (DORIGO, BIRATTARI & STÜTZLE, 2006)

Nessa configuração, as formigas que, pelo acaso, escolheram o caminho mais curto serão as primeiras a alcançar a fonte de alimento. A menor ponte receberá, então, feromônio mais rapidamente que a ponte maior e tal fato aumentará a probabilidade da menor ponte ser escolhida pelas demais formigas.

Essa mesma estratégia é utilizada pelas formigas para contornar um obstáculo pelo caminho mais curto, quando esse obstáculo passa a interromper o caminho previamente seguido por elas.

Outro fator que auxilia o processo de escolha do menor caminho é a evaporação do feromônio. Quando um caminho se torna obsoleto pelo esgotamento da fonte de alimento, a quantidade de feromônio que marca esse caminho é gradualmente reduzida pela evaporação, tornando-o cada vez menos atrativo. Isso constitui um processo de *feedback* negativo que ajuda as formigas a detectarem novos caminhos.

Dessa forma, as formigas convergem para um caminho mais curto sem comunicação direta, sem informação visual e sem ordem hierárquica. Tudo o que elas fazem é modificar o ambiente depositando feromônio, estabelecendo assim uma comunicação indireta que influenciará seus comportamentos levando a uma escolha bem definida do melhor caminho a ser seguido. Essa característica é uma propriedade desejável em muitos sistemas artificias.

3.2 A metaheurística *ACO*

Uma metaheurística é um *framework* algorítmico de propósito geral — ou um conjunto de conceitos algorítmicos que podem ser usadas para definir métodos heurísticos — que pode ser aplicado a diferentes problemas de otimização com relativamente poucas modificações.

A metaheurística *ACO*, formalizada em Dorigo, Di Caro e Gambardella (1999), pode ser descrita em suma, como se segue: em um algoritmo *ACO*, cada caminho seguido por uma formiga é associada a uma solução candidata para um dado problema e a quantidade de feromônio depositada sobre ele é proporcional à qualidade da solução candidata correspondente. Quanto maior a quantidade de feromônio presente em um caminho, maior a probabilidade do caminho em questão ser escolhido pela formiga.

Resultando dessa dinâmica, as formigas eventualmente convergem para um caminho mais curto, que pode se configurar na solução óptima ou quase óptima para o problema alvo.

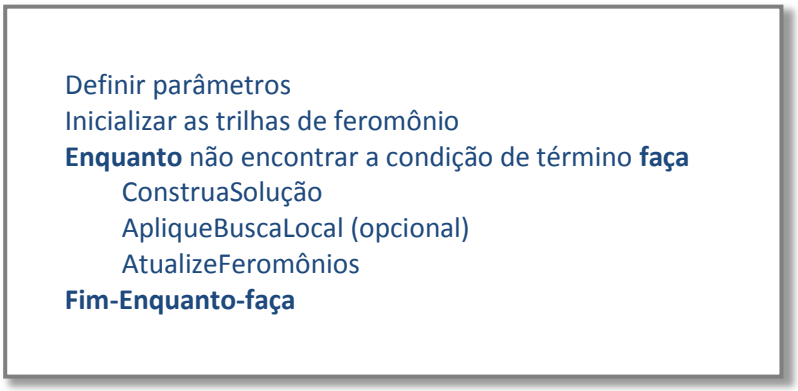

Figura 3.3 A metaheurística da colonização por colônia de formigas (DORIGO, BIRATTARI & STÜTZLE, 2006)

A Figura 3.3 mostra a metaheurística em questão. Após a definição dos parâmetros e inicialização das trilhas de feromônio, ocorre uma iteração em que primeiramente, as formigas constroem um certo número de soluções. Cada construção é iniciada por uma solução parcial vazia e em cada passo, um componente é adicionado. A escolha do componente é feita estocasticamente, priorizando os componentes que possuem uma maior quantidade de feromônio associada. Em seguida, essas soluções, são opcionalmente aprimoradas através de uma busca local. Por fim, o feromônio é atualizado de forma que as boas soluções tenham seu nível de feromônio aumentado e as más soluções tenham seu nível de feromônio reduzido, simulando a evaporação.

Um exemplo simples e bem sucedido de aplicação da metaheurística *ACO* é a otimização por colônia de formigas para a resolução do problema do caixeiro viajante. A introdução do problema é como se segue: dados um conjunto de cidades e a distância entre cada uma delas, o objetivo é encontrar um caminho hamiltoniano de comprimento mínimo em um grafo totalmente conectado, em outras palavras, o objetivo é encontrar o caminho mais curto que permite que cada cidade seja visitada uma única vez.

A estratégia para resolver o problema do caixeiro viajante utilizando a otimização por colônia de formigas é fazer com que as formigas caminhem sobre o grafo. Cada um dos vértices do grafo representa uma cidade e cada aresta representa uma conexão entre duas cidades tendo associada a ela uma variável feromônio que pode ser lida e alterada pelas formigas.

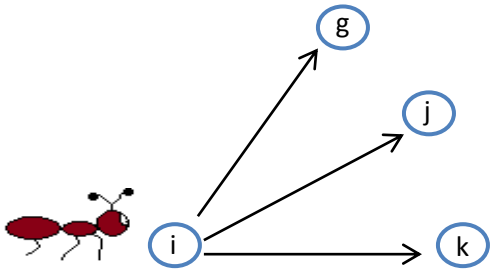

Figura 3.4 A formiga, da cidade i, escolhe a próxima cidade a ser visitada de maneira estocástica (DORIGO, BIRATTARI & STÜTZLE, 2006)

Cada uma das formigas constrói uma solução caminhando de um vértice a outro no grafo, com a restrição de não poder visitar um vértice que já tenho sido visitado em sua caminhada.

A Figura 3.4 representa uma etapa da construção da solução em que a formiga, localizada na cidade i, deve escolher o próximo vértice ou cidade a ser visitada. Tal escolha é feita estocasticamente entre os vértices ainda não visitados e a probabilidade que as cidades g, j e k sejam visitadas é proporcional à quantidade de feromônio associada, respectivamente, às arestas (i, g), (i, j) e (i, k).

Os valores do feromônio podem ser modificados no final de cada iteração, dependendo da qualidade das soluções construídas pelas formigas. Isso influenciará as iterações futuras a construir soluções semelhantes às melhores já construídas.

Capítulo 4 - Modelagem Fuzzy

A biomimética, o campo que estuda o desenvolvimento de produtos artificiais ou máquinas que imitam os fenômenos biológicos, tem sido vastamente explorada. Os fenômenos naturais apresentam características imprecisas e podem ser facilmente convertidos em uma descrição verbal. Por esse motivo, a modelagem fuzzy consiste em uma ferramenta adequada para abordar a biomimética.

Na modelagem *fuzzy*, inicialmente, o conhecimento é extraído de um especialista humano e convertido em uma coleção de regras SE-ENTÃO, utilizando a linguagem natural. A definição dos termos verbais, funções de pertinência e a inferência da base de regras, provenientes da teoria da lógica *fuzzy*, são os passos seguintes que levarão a geração do modelo matemático.

4.1 A Lógica *Fuzzy*

A teoria dos conjuntos nebulosos foi desenvolvida por Lotfi Zadeh em 1965. Sua meta era englobar aspectos vagos da informação. A teoria dos conjuntos clássica é em um caso particular da teoria dos conjuntos nebulosos.

A lógica *fuzzy* consiste na teoria dos conjuntos nebulosos aplicada em um contexto lógico, como o de sistemas baseados em conhecimento.

4.1.1 Função de pertinência

A definição formal da função de pertinência é dada por μ_A: $\Omega \rightarrow [0,1]$, onde A é um conjunto nebuloso do universo de discurso Ω. A função de pertinência mapeia cada elemento x de Ω com o grau $\mu_A(x)$, que é um número real no intervalo [0,1] e representa o grau de inclusão em que x pertence a A.

Se $\mu_A(x) = 0$, isso significa que x é completamente incompatível com o conceito expresso por A. Se $\mu_A(x) = 1$, x é completamente compatível com A. E, finalmente, se $0 < \mu_A(x) < 1$, x é parcialmente compatível com A, com o grau $\mu_A(x)$.

4.1.2 Controladores Nebulosos

Processos sofisticados e complexos podem ser eficientemente controlados por controladores nebulosos principalmente, quando o modelo matemático está sujeito a imprecisões e incertezas. Tais controladores são simples, robustos, versáteis, têm baixo custo e baixa manutenção.

Os controladores nebulosos, propostos em Mamdani (1976), são sistemas *fuzzy* compostos de um conjunto de regras de produção. Essas regras, que são do tipo SE-ENTÃO, determinam quais as ações de controle baseando-se nos valores das variáveis de entrada do problema. Tais valores, que nem sempre são bem definidos, são modelados por conjuntos nebulosos e constituem os chamados termos linguísticos.

A quádrupla (X, Ω, T(X), M) define o que chamamos de variável linguística. X representa o nome da variável, Ω o universo de discurso de X, T(X) o conjunto de rótulos dos valores de X e M denota a função de associação de cada elemento de T(X) a uma função de pertinência.

A Figura 4.1 exibe um modelo geral da estrutura básica de um controlador nebuloso. O primeiro módulo rotulado de “fuzificação” recebe as variáveis de

entrada, — também chamadas variáveis de estado —, que são escalares ou “crisp”, e as transforma em conjuntos nebulosos convertendo-as em instâncias da variável linguística.

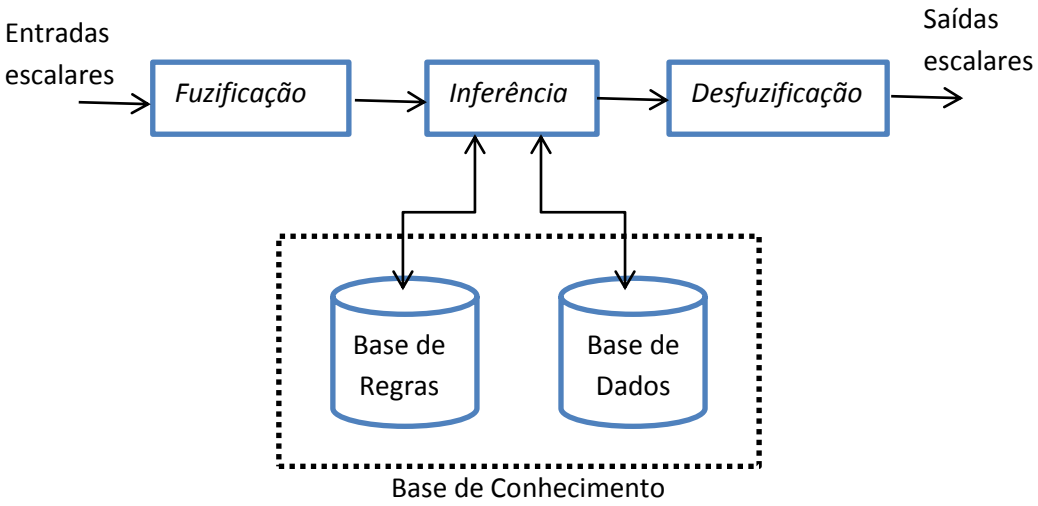

Figura 4.1 Estrutura básica de um controlador nebuloso (SANDRI & CORREA, 1999)

A base de conhecimento é composta pela base de regras e pela base de dados. A base de dados armazena as definições das funções de pertinência, bem como as definições de normalização e discretização dos universos de discurso. A base de regras possui o conjunto de regras de produção que apresentam a estrutura *SE <premissa> ENTÃO <consequente>.*

O módulo de inferência processa os dados de entrada e as regras de produção inferindo as ações de controle nebulosas de acordo com o estado do sistema.

O processo de desfuzificação associa o domínio das variáveis de saída a um universo de discurso e a partir da ação de controle nebulosa inferida no módulo anterior é definida uma ação de controle não nebulosa. O método de desfuzificação mais utilizado é chamado método do centro de gravidade. Nele, calcula-se a área da curva da variável linguística de saída produzida pelo processo de inferência, e acha-

se o índice correspondente que divide esta área pela metade, produzindo a saída: $D = \frac{\sum_{i=1}^{n} \mu.A(x_i).x_i}{\sum_{i=1}^{n} \mu.A(x_i)}$ onde “n” denota o número de níveis de quantização.

4.2 Modelando o comportamento das formigas

Apresentaremos nesta seção a modelagem do comportamento das formigas partindo de uma descrição verbal sobre esse comportamento e utilizando da teoria da lógica fuzzy para criar um modelo matemático que pode ser implementado por sistemas artificiais.

Esse experimento foi realizado em Rozine e Margaliot (2007) e apresentou como resultado um modelo matemático simples, congruente ao observado na natureza apresentando vantagens significativas sobre os demais modelos anteriormente sugeridos.

O processo mais específico a ser modelado é o de escolha do caminho de uma formiga frente a uma bifurcação. Utilizaremos a seguinte descrição verbal: “Se uma formiga se depara com uma bifurcação em uma trilha, quanto maior for a concentração de feromônio na ramificação da esquerda em relação à ramificação da direita, maior a probabilidade da ramificação da esquerda ser escolhida.”

Essa simples descrição verbal será transformada em um conjunto de regras fuzzy também bastante simples.

O modelo apresenta duas variáveis de entrada L e R que denotam a concentração de feromônio nas ramificações esquerda e direita, respectivamente. A variável de saída P = P(L, R), denota a probabilidade de escolha da ramificação da esquerda.

De acordo com a descrição verbal fornecida, a probabilidade de escolha da ramificação da esquerda é diretamente relacionada à diferença em concentração de feromônio D := L – R.

Usando a diferença D derivamos duas regras fuzzy:

- SE D é positiva ENTÃO P=1
- SE D é negativa ENTÃO P=0

Definiremos uma função de pertinência para o termo “positiva”, $\mu_{pos}(.)$, considerando que $\mu_{pos}(D)$ é uma função crescente, $\lim_{D\to-\infty} \mu_{pos} = 0$ e também $\lim_{D\to\infty} \mu_{pos} = 1$.

Há boas razões para utilizarmos a função tangente hiperbólica na modelagem fuzzy, assim utilizaremos a seguinte função de pertinência:

$$\mu_{pos}(D) := \frac{(1 + tanh(qD))}{2}.$$

O parâmetro q > 0 determina a curvatura de $\mu_{pos}(D)$. O experimento apresentou bons resultados para o valor q = 0,016. O termo “negativa” será modelado utilizando $\mu_{neg}(D) := 1 - \mu_{pos}(D)$.

Utilizando a inferência de centro da gravidade, obtemos:

$$P(D) = \mu_{pos}(D) / (\mu_{pos}(D) + \mu_{neg}(D))$$

$$P(D) = \frac{(1 + tanh(qD))}{2} \qquad \textit{(Equação 1)}$$

Sendo P(D) ∈(0,1) para todo D ∈ IR.

Capítulo 5 - O Algoritmo *Ant-Miner*

Neste capítulo apresentaremos o algoritmo *Ant-Miner* original proposto por Parpinelli, Lopes e Freitas (2002) que servirá de base para as modificações propostas no capítulo seguinte.

5.1 Descrição geral

O Algoritmo *Ant-Miner* é um algoritmo ACO cujo problema alvo é descobrir regras de classificação que possuem a forma: SE < termo1 E termo2 E …> ENTÃO < classe >.

Cada termo é uma tupla < atributo, operador, valor >, onde o operador é o "=", já que o *Ant-Miner* lida apenas com atributos categóricos e o valor pertence ao domínio de atributo.

A Figura 5.1 mostra uma descrição de alto nível do algoritmo *Ant-Miner*. O algoritmo segue a estratégia de cobertura, discutida no capítulo 2, descobrindo regras de classificação até que todos ou quase todos os casos de treinamento sejam cobertos.

Antes de o algoritmo iniciar sua execução, alguns parâmetros devem ser definidos pelo usuário. São eles: No_of_ants que determina o número de formigas, Min_cases_per_rule que expressa o número mínimo de casos que uma regra deve ser capaz de cobrir, Max_Uncovered_Cases que determina o número máximo de casos não cobertos no conjunto de treinamento, No_rules_converg que define o número de regras usadas para testar a convergência das formigas e Num_iterations que define o número máximo de iterações caso a convergência não ocorra.

```
TrainingSet = {todos os conjuntos de treinamento};
DiscoveredRuleList = [ ];   /* a lista de regras é inicializada com uma lista vazia */
ENQUANTO (TrainingSet > Max_uncovered_cases) FAÇA
      t = 1;   /* índice da formiga */
      j = 1;   /* índice do teste de convergência */
      Inicialize todas as trilhas com a mesma quantidade de feromônio;
      REPITA
            A formiga t inicia com uma regra vazia e constrói incrementalmente a regra
            de classificação Rt adicionando um termo de cada vez à regra atual;
            Pode a regra Rt;
            Atualize o feromônio de todas as trilhas aumentando o feromônio nas trilhas
            seguidas pela formiga t (proporcional à qualidade de Rt)e reduzindo o
            feromônio das outras trilhas (simulando a evaporação do feromônio)
            SE Rt é igual a Rt -1   /*atualize o teste de convergência*/
                  ENTÃO j = j + 1;
                  SENÃO j = 1;
            FIM-SE
            t = t + 1;
      ATÉ (i ≥ No_of_ants) OU (j ≥ No_rules_converg)
      Escolha a melhor regra Rmelhor entre todas as regras Rt construídas por todas as
      formigas;
      Adicione a regra Rmelhor à DiscoveredRuleList;
      TrainingSet = TrainingSet – {conjunto de casos cobertos corretamente por Rmelhor };
FIM-ENQUANTO
```

Figura 5.1 Descrição de alto nível do algoritmo *Ant-Miner* original (PARPINELLI, LOPES & FREITAS, 2002a)

Primeiramente, a variável TrainingSet é inicializada com todos os conjuntos de treinamento e a variável DiscoveredRuleList é inicializada com uma lista vazia. Cada iteração do laço ENQUANTO descobre uma regra de classificação que é adicionada a lista de regras descobertas DiscoveredRuleList e os casos de treinamento corretamente cobertos pela regra são removidos do conjunto de treinamento TrainingSet. O laço termina quando o número de casos do conjunto de treinamento for menor ou igual ao limite Max_uncovered_cases estipulado pelo usuário.

O laço REPITA-ATÉ inicia com a construção da regra. A formiga t começa com uma regra vazia e adiciona a ela um termo de cada vez. A regra parcial construída pela formiga corresponde ao caminho parcial percorrido por ela e o termo escolhido para ser adicionado corresponde à escolha da direção em que a formiga deve seguir. A adição de termos continua até que todos os atributos já tenham sido usados pela formiga ou até que a adição de um novo termo faça com que a regra passe a cobrir um número menor de casos que o estipulado pelo limite Min_cases_per_rule definido pelo usuário. Cada atributo deve aparecer somente uma vez na regra para evitar invalidações como “SE sexo = feminino E sexo = masculino”.

Em seguida, a regra R_t é podada para que sejam removidos termos irrelevantes e então, as concentrações de feromônio são atualizadas de forma que as trilhas percorridas pelas formigas recebam mais feromônio e as demais trilhas tenham suas taxas de feromônio reduzidas, simulando a evaporação. Utilizando essa nova configuração de feromônio, a próxima formiga inicia a construção de sua regra até que o número de regras construídas seja maior ou igual ao parâmetro No_of_ants ou até que a formiga t tenha construído a mesma regra que as No_rules_converg – 1 formigas anteriores, indicando a convergência das formigas.

Após o término do laço REPITA-ATÉ, a melhor regra dentre todas as construídas é adicionada a lista DicoveredRuleList e o laço ENQUANTO-FAÇA itera novamente.

Seja o $termo_{ij}$ da forma $A_i = V_{ij}$, onde A_i representa o i-ésimo atributo e V_{ij} é o j-ésimo valor do domínio de A_i. A probabilidade do $termo_{ij}$ ser adicionado à regra parcial é dada por:

$$P_{ij} = \frac{\eta_{ij} \cdot \tau_{ij}(t)}{\sum_{i=1}^{a} x_i \cdot \sum_{j=1}^{b_i} (\eta_{ij} \cdot \tau_{ij}(t))}$$

onde:

- η_{ij} é o valor de uma função heurística dependente do problema para o termo$_{ij}$. Quanto maior for seu valor, maior é a relevância do termo$_{ij}$ e a probabilidade dele ser escolhido.

- $\tau_{ij}(t)$ é a quantidade de feromônio associada ao termo$_{ij}$ na iteração t.

- a é o número total de atributos.

- x_i é um parâmetro que controla a unicidade do uso do atributo A_i. x_i recebe 1 se o atributo ainda não foi usado pela formiga atual e recebe 0 caso contrário.

- b_i é o número de valores do domínio do i-ésimo atributo.

Depois de terminada a construção do antecedente da regra, o sistema escolhe a classe que maximiza a qualidade da regra. Na prática, a classe que aparece mais vezes entre os casos cobertos pela regra será escolhida.

5.2 Função heurística

A função heurística η_{ij} é uma estimativa da qualidade do termo$_{ij}$, que exprime o quanto a sua adição permite melhorar a acurácia preditiva da regra. Essa medida é chamada de entropia. A entropia de um termo$_{ij}$ da forma $A_i=V_{ij}$ é dada por:

$$H(W|A_i = V_{ij}) = -\sum_{w=1}^{k} \left(P(w|A_i = V_{ij}\right) . \log_2\left(P(w|A_i = V_{ij})\right)$$

onde W é atributo classe, k é o número de valores pertencentes ao domínio do atributo classe e P(w | $A_i = V_{ij}$) é a probabilidade empírica de se observar a classe w, tendo observado o termo $A_i = V_{ij}$.

Por exemplo, utilizando os dados da Tabela 2.1, vamos calcular a entropia do termo “Aspecto = ensolarado”:

$$H(Jogar \mid Aspecto = ensolarado)$$

$$= -\sum_{w=1}^{2} (P(w|Aspecto = ensolarado)\ .\ \log_2(P(w|Aspecto = ensolarado))$$

A probabilidade empírica do referido termo para a classe “sim” é igual a 2/5 e para a classe “não” é igual a 3/5. Logo,

$$H(Jogar \mid Aspecto = ensolarado)$$
$$= -(2/5 \log_2 2/5 + \ 3/5 \log_2 3/5) = \ 0{,}9709508$$

Para o termo “Humidade = normal”, que aparece 6 vezes com a classe “sim” e uma vez com a classe “não”, a entropia é:

$$H(Jogar \mid Humidade = normal)$$
$$= -(6/7 \log_2 6/7 + \ 1/7 \ \log_2 1/7) = \ 0{,}591673$$

Quanto maior o valor da entropia, mais uniforme é a distribuição das classes e, portanto, menor a probabilidade do termo ser escolhido.

A função heurística é normalizada para facilitar seu uso e assim, temos:

$$\eta_{ij} = \frac{\log_2 k - H(W|A_i = V_{ij})}{\sum_{i=1}^{a} x_i\ .\ \sum_{j=1}^{b_i}(\log_2 k - H(W|A_i = V_{ij}))}$$

Antes de aplicar a fórmula da função heurística, duas situações devem ser checadas. Se não houver nenhuma ocorrência de um determinado termo no conjunto de treinamento, o valor da entropia do termo em questão será definido com o valor máximo $\log_2 k$. Se todas as ocorrências de um termo no conjunto de treinamento forem associadas a uma única classe, a entropia do termo será definida como zero, atribuindo assim, a maior capacidade de predição ao termo, o que aumentará a sua probabilidade de ser escolhido.

A função da entropia é uma heurística local que considera um atributo de cada vez. Por outro lado, a atualização do feromônio considera as interações entre os atributos, já que considera a performance da regra como um todo. Para evitar soluções ótimas locais, a entropia é usada juntamente com a atualização do feromônio.

A utilização da função heurística se faz importante, principalmente, no início da criação de cada regra, quando todas as trilhas possuem a mesma concentração de feromônio.

5.3 Atualização do feromônio

A configuração inicial do feromônio, quando todas as trilhas recebem a mesma quantidade da substância, é inversamente proporcional à quantidade de valores no domínio de todos os atributo. A concentração de feromônio inicial do $termo_{ij}$ é dada por:

$$\tau_{ij}(t=0) = \frac{1}{\sum_{i=1}^{a} b_i}$$

onde a é o número de atributos e b_i é o número de valores no domínio do atributo A_i.

Após a construção e poda da regra, o feromônio de cada termo que a constitui deve ser incrementado proporcionalmente à qualidade da regra. Os demais termos terão sua concentração de feromônio reduzida, simulando a evaporação.

A qualidade Q da regra é dada pela fórmula Q = sensibilidade × especificidade, definida como:

$$Q = \frac{VP}{VP + FN} \cdot \frac{VN}{FP + VN}$$

onde:

- VP significa verdadeiros positivos e contabiliza o número de casos cobertos pela regra que têm a mesma classe predita pela regra.

- FP significa falsos positivos e contabiliza o número de casos cobertos pela regra que têm uma classe diferente da predita pela regra.

- FN significa falsos negativos e contabiliza o número de casos que não são cobertos pela regra, mas que têm a mesma classe predita pela regra.

- VN significa verdadeiros negativos e contabiliza o número de casos que não são cobertos pela regra e possuem uma classe diferente da predita pela regra.

Quanto maior o valor de Q, maior a qualidade da regra. O valor de Q está compreendido entre 0 e 1.

A atualização do feromônio do termo$_{ij}$ é dada pela equação:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \tau_{ij}(t).Q, \forall_{i,j} \in R$$

onde R é o conjunto de termos presentes na regra construída pela formiga na iteração t.

A redução do feromônio é feita dividindo a quantidade atual de feromônio de um dado termo não usado pela soma total de feromônio de todos os termos que tiveram sua taxa de feromônio incrementada, resultando na redução da quantidade de feromônio normalizada para cada termo não usado.

5.4 Poda das regras

A poda de regras é uma prática bastante utilizada em mineração de dados, seu objetivo principal é remover termos irrelevantes, aumentando o poder preditivo da regra, evitando o *overfitting* com os dados de treinamento e aumentando sua simplicidade.

A poda é realizada logo após a construção da regra e consiste em remover um termo de cada vez, recalculando a qualidade da regra sem cada termo. O termo cuja remoção acarrete em uma melhora mais significativa da qualidade da regra é removido e esse processo é iterado até que a regra tenha somente um termo ou até que não exista mais nenhum termo cuja remoção melhore a qualidade da regra.

Capítulo 6 - Modificações propostas

Conforme apresentado no capítulo 4, a modelagem fuzzy do comportamento das formigas frente a uma bifurcação — partindo da descrição verbal: “Se uma formiga se depara com uma bifurcação em uma trilha, quanto maior for a concentração de feromônio na ramificação da esquerda em relação à ramificação da direita, maior a probabilidade da ramificação da esquerda ser escolhida” — gerou a seguinte fórmula da probabilidade de escolha do caminho da esquerda:

$$P(D) = \frac{(1 + tanh(qD))}{2}$$

(*Equação 1*)

onde D representa a diferença de concentração de feromônio na ramificação esquerda menos a concentração de feromônio na ramificação direita, denotado por $D = L - R$ e q é o parâmetro que determina a inclinação da curva. Adotaremos também para nossos experimentos o valor de $q = 0{,}016$, conforme em Rozine e Margaliot (2007).

Nosso objetivo principal é adaptar a *Equação 1* ao algoritmo *Ant-Miner*. Para isso, precisamos iniciar definindo D.

6.1 Estratégia local e global

Iremos denotar L por $\tau_{ij}(t)$, que pode ser lido como a concentração de feromônio do termo$_{ij}$ na iteração t.

Vamos considerar duas possibilidades para as opções de escolha do caminho a ser seguido pela formiga:

a) Primeiro, buscando uma suposta simplicidade, vamos imaginar que a probabilidade da formiga escolher um determinado termo$_{ij}$ dependa apenas da diferença de concentração de feromônio entre o termo$_{ij}$ e os demais valores do domínio do atributo i. Assim, $R = (\sum_{j=1}^{b_i} \tau_{ij}(t)) - \tau_{ij}(t)$, sendo b_i o número total de valores no domínio do atributo i. Dessa forma, temos:

$$D = 2\,\tau_{ij}(t) - \sum_{j=1}^{b_i} \tau_{ij}(t)$$

Chamaremos esta interpretação de estratégia local. Sempre que a estratégia local for aplicada, a configuração inicial do feromônio também deve ser redefinida para que somente os valores pertencentes ao domínio do atributo i sejam considerados. Assim, a concentração inicial de feromônio do termo$_{ij}$ passa a ser dada por:

$$\tau_{ij}(t = 0) = \frac{1}{b_i}$$

b) Em segundo lugar, para evitar que a formiga tenha uma visão limitada e acabe por encontrar soluções ótimas somente locais, vamos considerar que a probabilidade de escolha do termo$_{ij}$ seja baseada na diferença de concentração de feromônio entre o termo$_{ij}$ e todos os demais valores do domínio de todos os atributos que ainda não foram utilizados pela formiga. Assim, $R = (\sum_{i=1}^{a} x_i * \sum_{j=1}^{b_i} \tau_{ij}(t)) - \tau_{ij}(t)$, sendo a o número total de atributos e x_i um parâmetro que controla se o atributo já foi utilizado (quando lhe é atribuído o valor 0) ou não ($x_i = 1$). Dessa forma, obtemos:

$$D = 2\,\tau_{ij}(t) - \sum_{i=1}^{a} x_i * \sum_{j=1}^{b_i} \tau_{ij}(t)$$

Com base nessas duas interpretações, podemos derivar da *Equação 1* duas fórmulas para a probabilidade de escolha do termo$_{ij}$:

a) $$P_{ij} = \frac{1+\tanh(q\ (2\ \tau_{ij}(t) - \sum_{j=1}^{b_i} \tau_{ij}(t)))}{b_i}$$ (*Equação 1a*)

b) $$P_{ij} = \frac{1+\tanh(q\ (2\ \tau_{ij}(t) - \sum_{i=1}^{a} x_i * \sum_{j=1}^{b_i} \tau_{ij}(t)))}{\sum_{i=1}^{a} x_i * b_i}$$ (*Equação 1b*)

A fórmula original da probabilidade do *Ant-Miner* foi substituída pela *Equação 1a* na versão que chamamos de *Ant-Miner 1a* e pela *Equação 1b* na versão que será referida como *Ant-Miner 1b*.

6.2 Função Heurística

As versões criadas anteriormente não fazem uso da informação heurística. Pelas vantagens previamente citadas, é desejável incorporar a função heurística às fórmulas da probabilidade. Para isso, vamos considerar que a diferença D seja dada pela diferença do produto da concentração de feromônio e da função heurística (η_{ij}). Logo:

a) $$D = 2\ \tau_{ij}(t) * \eta_{ij} - \sum_{j=1}^{b_i} \tau_{ij}(t) * \eta_{ij}$$

b) $$D = 2\ \tau_{ij}(t) * \eta_{ij} - \sum_{i=1}^{a} x_i * \sum_{j=1}^{b_i} \tau_{ij}(t) * \eta_{ij}$$

Daí obtemos duas novas fórmulas para a probabilidade que levam em conta a função heurística:

a) $$P_{ij} = \frac{1+\tanh(q\ (2\ \tau_{ij}(t)*\eta_{ij} - \sum_{j=1}^{b_i}(\tau_{ij}(t)*\eta_{ij})))}{b_i}$$ (*Equação 2a*)

b) $$P_{ij} = \frac{1+\tanh(q\ (2\ \tau_{ij}(t)*\eta_{ij} - \sum_{i=1}^{a} x_i * \sum_{j=1}^{b_i}(\tau_{ij}(t)*\eta_{ij})))}{\sum_{i=1}^{a} x_i * b_i}$$ (*Equação 2b*)

A implementação da *Equação 2a* deu origem à versão *Ant-Miner 2a* que utiliza a estratégia local e a implementação da *Equação 2b* originou a *versão Ant-Miner 2b* que utiliza a estratégia global.

6.3 Prioridade aos atributos utilizados

Na tentativa de aprimorar os modelos criados, duas observações foram feitas que levaram ao desenvolvimento das próximas modificações a serem aplicadas.

Primeiramente, nos remetendo a outros métodos de aprendizagem de máquina, pôde ser observado que em uma estratégia de divisão e conquista, a expansão da árvore ocorre para todos os valores de um atributo. Essa característica pode ser desejável, uma vez que, intuitivamente, pode ser interessante explorar mais do que um valor de um atributo. Por exemplo, se existe uma regra que diz que uma idade avançada pode levar a um alto risco de se desenvolver uma certa doença, é natural que exista outra regra que diga que um jovem terá um risco menor de desenvolver a mesma doença.

Voltando ao Algoritmo *Ant-Miner* da Figura 5.1, também foi observado que toda vez que o laço ENQUANTO-FAÇA é reiniciado, nenhuma informação das iterações anteriores que influencie a fórmula da probabilidade é mantida pelas formigas.

```
TrainingSet = {todos os conjuntos de treinamento};
DiscoveredRuleList = [ ];   /* a lista de regras é inicializada com uma lista vazia */
PARA CADA atributo i FAÇA
      fator_de_atracao_i = 0;
FIM-PARA
ENQUANTO (TrainingSet > Max_uncovered_cases) FAÇA
      t = 1;  /* índice da formiga */
      j = 1;   /* índice do teste de convergência */
      Inicialize todas as trilhas com a mesma quantidade de feromônio;
      REPITA
            A formiga t inicia com uma regra vazia e constrói incrementalmente a regra
            de classificação R_t adicionando um termo de cada vez à regra atual;
            Pode a regra R_t;
            Atualize o feromônio de todas as trilhas aumentando o feromônio nas trilhas
            seguidas pela formiga t (proporcional à qualidade de R_t)e reduzindo o
            feromônio das outras trilhas (simulando a evaporação do feromônio)
            SE R_t é igual a R_t -1   /*atualize o teste de convergência*/
                  ENTÃO j = j + 1;
                  SENÃO j = 1;
            FIM-SE
            t = t + 1;
      ATÉ (i ≥ No_of_ants) OU (j ≥ No_rules_converg)
      Escolha a melhor regra R_melhor entre todas as regras R_t construídas por todas as
      formigas;
      PARA CADA atributo i pertencente à R_melhor FAÇA
            SE valor j não aparece em regras anteriores ENTÃO
                fator_de_atracao_i = fator_de_atracao_i + 1 / numValoresAtributo_i;
            FIM-SE
      FIM-PARA
      Adicione a regra R_melhor à DiscoveredRuleList;
      TrainingSet = TrainingSet – {conjunto de casos cobertos corretamente por R_melhor };
FIM-ENQUANTO
```

Figura 6.1 Descrição de alto nível do algoritmo *Ant-Miner* modificado

Pensando nessas duas situações, resolvemos agregar uma certa vantagem aos termos cujo atributo já tenha sido utilizado em uma regra anterior. Inicialmente, a variável fator_de_atracao foi criada para cada atributo e inicializada com zero. Após o término do laco REPITA-ATÉ, que constrói as regras candidatas, e da definição da melhor regra R_{melhor}, cada atributo i utilizado em R_{melhor} tem sua variável fator_de_atracao atualizada se o valor j que faz par com o atributo i não tiver sido utilizado anteriormente nas regras de DiscoveredRuleList, conforme:

$$fator\_de\_atracao_i = fator\_de\_atracao_i + 1 / numValoresAtributo_i$$

onde $numValoresAtributo_i$ é o número de valores pertencentes ao domínio do atributo i.

Essa modificação pode ser observada na Figura 6.1. Nas funções de cálculo da probabilidade anteriormente criadas foi adicionado um teste que verifica se a variável fator_de_atracao do atributo é igual a um ou maior do que zero e menor que um.

No primeiro caso, se o teste for positivo, isso significa que todos os valores do domínio do atributo já foram utilizados e, como ainda pode ser interessante que esse atributo continue sendo explorado em regras futuras, a probabilidade da escolha dos valores pertencentes ao domínio do atributo foi dobrada.

No segundo caso, quando nem todos os valores do domínio do atributo foram utilizados, a probabilidade dos referidos valores foi triplicada, para que as chances sejam ainda maiores de que todos os valores do atributo sejam explorados. Nenhuma alteração foi aplicada para os casos em que o fator_de_atracao é igual a zero.

Não apresentamos garantias de que a multiplicação da probabilidade por 2 e por 3 sejam valores óptimos. Esses valores foram utilizados por apresentarem bons resultados em nossos experimentos preliminares. A otimização desses parâmetros deve ser explorada em pesquisas futuras.

Esse recurso que prioriza os atributos já utilizados foi implementado para as quatro versões do *Ant-Miner* criadas anteriormente e serão referidas neste trabalho como *Ant-Miner 1a Attribute Priority*, *Ant-Miner 1b Attribute Priority, Ant-Miner 2a Attribute Priority* e *Ant-Miner 2b Attribute Priority*.

Capítulo 7 - Experimentos e análise dos resultados

Para a realização dos experimentos, sete *data sets* de domínio público do repositório da Universidade da Califórnia foram utilizados, sendo eles: *Ljubjana Breast Cancer, Wisconsin Breast Cancer, Tic-tac-toe, Dermatology, Hepatitis, Cleveland Heart Disease* e *Diabetes.* A Tabela 7.1 apresenta informações sobre os *data sets* utilizados.

Data Set	Nº de casos	Nº de atributos categóricos	Nº de atributos contínuos	Nº de classes
Ljubljana	286	9	-	2
Wisconsin	699	-	9	2
Tic-tac-toe	958	9	-	2
Dermatology	366	33	1	6
Hepatitis	155	13	6	2
Cleveland	303	7	6	5
Diabetes	768	-	8	2

Tabela 7.1 Dados sobre os *data sets* utilizados no experimento

Pelo fato do algoritmo *Ant-Miner* não ter suporte à atributos contínuos, foi necessário, durante o pré-processamento dos dados, fazer a discretização de todos os atributos numéricos. Para isso, foi utilizado o filtro de discretização não supervisionada do *Weka*, que permite ao usuário pré-definir o número de intervalos nos quais os dados serão divididos. A divisão foi estabelecida de forma a distribuir os dados o mais uniformemente possível.

Por se tratar de um experimento estocástico, a coleta dos resultados foi realizada executando cada versão criada do programa dez vezes com cada data set, utilizando validação cruzada de dez *folds*. Ao final, foram calculadas as médias aritméticas da acurácia obtida em cada uma das execuções.

Conforme mencionado no capítulo 5, alguns parâmetros devem ser configurados antes da execução do algoritmo. Em nosso primeiro experimento, os parâmetros foram definidos como:

- No_of_ants: 10;
- Min_cases_per_rule: 5;
- Max_uncovered_cases: 10;
- No_rules_converg: 10;
- Num_iterations: 100.

Primeiramente, foram analisadas as versões original, *Ant-Miner 1a*, *Ant-Miner 1b, Ant-Miner 2a* e *Ant-Miner 2b*. Os resultados são exibidos na Tabela 7.2.

Data Set	Ant-Miner	AM 1a	AM 1b	AM 2a	AM 2b
Ljubljana	73,60% +/- 2,57%	74,71% +/- 2,42%	75,00% +/- 3,15%	74,38% +/- 3,09%	74,56% +/- 2,46%
Wisconsin	91,85% +/- 1,00%	91,84% +/- 0,92%	92,02% +/- 1,17%	92,41% +/- 1,10%	92,49% +/- 0,90%
Tic-tac-toe	72,11% +/- 1,86%	69,65% +/- 1,91%	70,25% +/- 1,69%	69,53% +/- 1,76%	70,11% +/- 1,52%
Dermatology	95,28% +/- 1,18%	95,79% +/- 1,63%	95,32% +/- 1,15%	95,53% +/- 0,91%	95,62% +/- 0,97%
Hepatitis	83,67% +/- 2,91%	77,11% +/- 3,19%	79,17% +/- 3,51%	79,00% +/- 3,26%	78,96% +/- 3,32%
Cleveland	77,99%+/-2,38%	76,06% +/- 2,61%	78,06% +/- 2,67%	78,86% +/- 1,99%	77,99% +/- 2,24%
Diabetes	67,16%+/-1,72%	66,26% +/- 1,85%	67,33% +/- 1,46%	66,32% +/- 1,67%	67,36% +/- 1,82%

Tabela 7.2 Resultado da acurácia preditiva das versões original, 1a, 1b, 2a e 2b, utilizando Min_cases_per_rule=5 e Max_uncovered_cases=10

Alterando os parâmetros, testamos novamente as mesmas versões, obtendo os resultados da Tabela 7.3.

Nova configuração de parâmetros:

- No_of_ants: 10;
- Min_cases_per_rule: 10;
- Max_uncovered_cases: 10;
- No_rules_converg: 10;
- Num_iterations: 100.

Data Set	Ant-Miner	AM 1a	AM 1b	AM 2a	AM 2b
Ljubljana	74,98% +/- 2,75%	72,02% +/- 2,38%	75,05% +/- 2,93%	72,74% +/- 3,06%	75,28% +/- 2,74%
Wisconsin	92,82% +/- 0,97%	93,18% +/- 1,27%	93,02% +/- 0,97%	92,41% +/- 1,19%	93,34% +/- 0,83%
Tic-tac-toe	72,43% +/- 1,66%	69,38% +/- 1,10%	68,90% +/- 1,70%	69,35% +/- 1,62%	68,89% +/- 1,54%
Dermatology	95,34% +/- 1,04%	94,23% +/- 1,93%	95,53% +/- 1,01%	95,15% +/- 1,18%	95,14% +/- 1,18%
Hepatitis	83,21% +/- 3,18%	75,54% +/- 2,72%	78,91% +/- 3,06%	78,03% +/- 2,29%	78,06% +/- 3,14%
Cleveland	79,14% +/- 2,61%	77,43% +/- 1,82%	78,56% +/- 2,33%	78,54% +/- 1,91%	78,99% +/- 2,31%
Diabetes	67,67% +/- 1,73%	68,77% +/- 1,65%	67,28% +/- 1,69%	68,21% +/- 1,13%	67,28% +/- 1,61%

Tabela 7.3 Resultado da acurácia preditiva das versões original, 1a, 1b, 2a e 2b, utilizando Min_cases_per_rule=10 e Max_uncovered_cases=10

7.1 Comparação com a versão original

No primeiro experimento, a versão *Ant-Miner 1a* foi superior ao *Ant-Miner* original em apenas 2 *data sets*, dos 7 testados. A versão *Ant-Miner 2a* foi superior em 4, e as demais versões foram superiores em 5 dos *data sets*.

No segundo experimento, a versão *Ant-Miner 2a* foi superior à versão original em apenas um *data set*, as versões *Ant-Miner 2b* e *Ant-Miner 1a* foram superiores em 2 *data sets* e a versão *Ant-Miner 1b* foi superior em 3 dos *data sets*.

A versão *Ant-Miner* original com a nova configuração de parâmetros foi superior à mesma versão com a configuração de parâmetros anterior em 6 *data sets*. Isso indica que o *Ant-Miner* original produz melhores resultados criando regras mais gerais, que abranjam mais casos, até certo ponto.

Ao contrário, e exceto pela versão *Ant-Miner 1b*, as novas versões foram superiores a elas mesmas utilizando a configuração de parâmetros em que Min_cases_per_rule = 5, sendo as versões *Ant-Miner 1a* e *Ant-Miner 2b* superiores em 4 *data sets* e a versão *Ant-Miner 2a* superior em 6 *data sets*. A versão *Ant-Miner*

1b utilizando Min_cases_per_rule = 10 foi superior a ela mesma utilizando Min_cases_per_rule = 5 em 4 *data sets*.

A maior variação ocorreu na versão *Ant-Miner 2a*, o que nos leva a crer que a estratégia local combinada à função heurística beneficia a criação de regras mais específicas.

O equilíbrio das variações apresentado nas versões *Ant-Miner 1a, Ant-Miner 1b* e *Ant-Miner 2b* demonstra que as essas versões são menos sensíveis à variação do parâmetro Min_cases_per_rule.

7.2 Análise da estratégia local e global

Comparando as versões *Ant-Miner 1a* e *Ant-Miner 1b*, podemos notar que a estratégia global da versão *Ant-Miner 1b* apresentou uma vantagem significativa com relação à estratégia local da versão *Ant-Miner 1a*. A visão limitada da formiga para o cálculo da probabilidade sem a utilização de nenhum outro artifício fez com que o algoritmo convergisse para uma solução ótima local, o que gerou resultados piores aos comparados com a versão *Ant-Miner 1b* que utiliza a estratégia global.

7.2.1 Função heurística

Quando combinadas à função heurística, a estratégia global continuou superior à estratégia local. A versão *Ant-Miner 2b* foi superior à *Ant-Miner 2a* em 5 *data sets* no primeiro experimento e em 4 no segundo.

Em geral, as versões que apresentaram melhor desempenho foram as versões que utilizam a estratégia global, sendo a versão *Ant-Miner 1b* melhor que as demais no primeiro experimento e a versão *Ant-Miner 2b* melhor que as demais, exceto pela versão original, no segundo experimento. Tal resultado indica que a função heurística combinada à estratégia global beneficia ligeiramente a criação de regras mais gerais.

7.3 Análise da prioridade ao atributo utilizado

Nosso terceiro experimento envolve testar as versões original, *Ant-Miner 1a Attribute Priority, Ant-Miner 1b Attribute Priority, Ant-Miner 2a Attribute Priority* e *Ant-Miner 2b Attribute Priority.*

Os parâmetros foram configurados da seguinte forma:

- No_of_ants: 10;
- Min_cases_per_rule: 5;
- Max_uncovered_cases: 10;
- No_rules_converg: 10;
- Num_iterations: 100.

Os resultados são mostrados na Tabela 7.4.

Nosso quarto experimento foi realizado alterando os parâmetros para:

- No_of_ants: 10;
- Min_cases_per_rule: 10;
- Max_uncovered_cases: 10;
- No_rules_converg: 10;
- Num_iterations: 100.

Data Set	Ant-Miner	AM 1a AP	AM 1b AP	AM 2a AP	AM 2b AP
Ljubljana	73,60% +/- 2,57%	75,04% +/- 1,86%	75,10% +/- 2,80%	76,63% +/- 3,21%	74,04% +/- 2,56%
Wisconsin	91,85% +/- 1,00%	92,92% +/- 0,95%	92,46% +/- 0,93%	91,90% +/- 0,92%	92,40% +/- 0,95%
Tic-tac-toe	72,11% +/- 1,86%	72,03% +/- 1,10%	70,21% +/- 1,70%	73,41% +/- 1,31%	70,41% +/- 1,38%
Dermatology	95,28% +/- 1,18%	95,79% +/- 0,95%	95,53% +/- 1,03%	95,84% +/- 1,34%	95,45% +/- 1,09%
Hepatitis	83,67% +/- 2,91%	80,01% +/- 3,25%	77,78% +/- 3,10%	78,41% +/- 4,02%	78,65% +/- 3,75%
Cleveland	77,99%+/-2,38%	77,41% +/- 1,58%	78,25% +/- 2,56%	78,09% +/- 2,32%	78,16% +/- 2,23%
Diabetes	67,16%+/-1,72%	66,48% +/- 1,58%	67,23% +/- 1,82%	68,69% +/- 1,65%	67,74% +/- 1,82%

Tabela 7.4 Resultado da acurácia preditiva das versões original, 1a AP, 1b AP, 2a AP e 2b AP, utilizando Min_cases_per_rule=5 e Max_uncovered_cases=10

As mesmas versões do terceiro experimento foram testadas e os resultados são exibidos na Tabela 7.5

Data Set	Ant-Miner	AM 1a AP	AM 1b AP	AM 2a AP	AM 2b AP
Ljubljana	74,98% +/- 2,75%	74,36% +/- 3,67%	74,25% +/- 2,53%	75,13% +/- 2,17%	74,36% +/- 2,64%
Wisconsin	92,82% +/- 0,97%	92,77% +/- 1,37%	93,04% +/- 0,99%	93,12% +/- 0,88%	93,11% +/- 1,05%
Tic-tac-toe	72,43% +/- 1,66%	71,59% +/- 2,08%	70,28% +/- 1,48%	72,79% +/- 0,76%	69,30% +/- 1,45%
Dermatology	95,34% +/- 1,04%	95,76% +/- 1,22%	95,67% +/- 0,99%	96,02% +/- 1,17%	95,64% +/- 1,01%
Hepatitis	83,21% +/- 3,18%	76,47% +/- 2,88%	78,58% +/- 3,25%	80,90% +/- 3,17%	77,61% +/- 3,62%
Cleveland	79,14% +/- 2,61%	78,46% +/- 2,07%	78,46% +/- 2,53%	78,59% +/- 2,31%	78,05% +/- 2,65%
Diabetes	67,67% +/- 1,73%	68,90% +/- 1,28%	67,78% +/- 1,61%	68,25% +/- 2,10%	67,37% +/- 1,56%

Tabela 7.5 Resultado da acurácia preditiva das versões original, 1a AP, 1b AP, 2a AP e 2b AP, utilizando Min_cases_per_rule=10 e Max_uncovered_cases=10

No terceiro experimento, as versão *Ant-Miner 2a Attribute Priority* foi superior ao *Ant-Miner* original em 6 dos *data sets* testados, enquanto as versões *Ant-Miner 1b Attribute Priority* e *Ant-Miner 2b Attribute Priority* foram superiores em 5 *data sets* e a versão *Ant-Miner 1a Attribute Priority* foi superior em 3.

No quarto experimento, a versão *Ant-Miner 2a Attribute Priority* foi superior à versão original em 5 dos *data sets*, a versão *Ant-Miner 1b Attribute Priority* foi

superior em 3 e as versões *Ant-Miner 2b Attribute Priority* e *Ant-Miner 1a Attribute Priority* em 2 *data sets*.

Comparando as versões *Attribute Priority* a elas mesmas nas duas diferentes configurações de parâmetros, 3 delas (*Ant-Miner 2a Attribute Priority, Ant-Miner 1b Attribute Priority* e *Ant-Miner 2b Attribute Priority*) foram superiores utilizando Min_cases_per_rule = 10 em 4, 6 e 4 *data sets* respectivamente. A versão *Ant-Miner 1a Attribute Priority* foi superior utilizando Min_cases_per_rule = 5 em 5 *data sets*.

A adição da prioridade ao atributo utilizado às versões *Ant-Miner 1a, Ant-Miner 2a e Ant-Miner 1b* em ambas as configurações de parâmetros aprimorou o desempenho dos algoritmos citados. Já a versão *Ant-Miner 2b* foi superior à versão *Ant-Miner 2b Attribute Priority* em 4 data sets nas duas configurações de parâmetros.

Analisando a estratégia local e global em conjunto com a prioridade ao atributo utilizado, foi verificado que a versão *Ant-Miner 1a Attribute Priority* foi superior à versão *Ant-Miner 1b Attribute Priority* em 4 *data sets* no terceiro experimento e em 5 *data sets* no quarto experimento, indicando que a visão limitada da formiga foi compensada pela prioridade atribuída ao atributo.

A versão *Ant-Miner 2a Attribute Priority* foi superior à versão *Ant-Miner 2b Attribute Priority* em 4 *data sets* no terceiro experimento e em 7 no quarto, o que demonstra que o recurso de atribuição da prioridade ao atributo se relaciona melhor com a estratégia local.

Entre as versões *Ant-Miner 1a Attribute Priority* e *Ant-Miner 2a Attribute Priority*, a segunda versão se mostrou superior à primeira em 5 *data sets* no teceiro experimento e em 6 *data sets* no quarto experimento.

Em geral, a versão *Ant-Miner 2a Attribute Priority* obteve um melhor desempenho que todas as outras incluindo o algoritmo *Ant-Miner* original. Além

disso, essa versão se mostrou mais robusta, sendo menos sensível a variação do parâmetro Min_cases_per_rule.

Capítulo 8 - Conclusões e Trabalhos Futuros

Inicialmente, no primeiro experimento, a aplicação da modelagem *fuzzy* utilizando a estratégia global gerou resultados superiores ao Ant-Miner original em termos de acurácia preditiva. Com a mudança de cenário, em que o parâmetro Min_cases_per_rule foi configurado para extrair regras mais gerais, o desempenho das novas versões caiu consideravelmente, sendo que nenhuma apresentou resultados superiores ao *Ant-Miner* original em mais de três *data sets*.

Analisando os resultados do primeiro e segundo experimento, pudemos confirmar que a modelagem *fuzzy* utilizando somente a estratégia global é superior a modelagem *fuzzy* utilizando somente a estratégia local, uma vez que a estratégia local, quando não combinada a nenhum outro recurso, faz com que o algoritmo convirja para soluções ótimas locais.

A incorporação da função heurística à modelagem *fuzzy* com estratégia global fez-se importante principalmente na tentativa de obtenção de regras mais gerais. Também nas novas versões criadas, cada primeira regra candidata gerada, obtida em cada primeira iteração do laço REPITA-ATÉ, é extraída com base apenas na informação heurística. Assim, nas versões que não utilizam a função heurística, a primeira regra candidata foi gerada de forma totalmente aleatória, prejudicando o fluxo contínuo de geração de regras de boa qualidade.

No terceiro e quarto experimentos, a aplicação da prioridade ao atributo utilizado às versões anteriormente criadas apresentou uma melhora em todas as versões, exceto pela versão *Ant-Miner 2b*. A incorporação dessa técnica apresentou-se mais efetiva quando combinada à modelagem *fuzzy* utilizando a estratégia local. Isso ocorre pelo fato de que a atribuição da prioridade ao atributo utilizado é capaz de compensar a visão limitada da formiga ao selecionar os termos constituintes das regras.

Comparando as oito versões criadas entre si nos diferentes cenários e cada uma delas com o algoritmo Ant-Miner original, a versão que apresentou o melhor desempenho foi a versão *Ant-Miner 2a Attribute Priority*, que utiliza a modelagem fuzzy com estratégia local, a técnica de priorização ao atributo utilizado e a função heurística. Tais elementos combinados se mostraram bastantes efetivos em diferentes configurações do sistema. Ao alterar o parâmetro Min_cases_per_rule, a variação obtida foi menor que a variação apresentada pelo algoritmo Ant-Miner original o que demonstra a robustez da versão *Ant-Miner 2a Attribute Priority.*

Futuras pesquisas nesta área podem ser orientadas à otimização dos parâmetros que definem a prioridade dada ao atributo utilizado. Outra intervenção interessante seria determinar a configuração ótima dos parâmetros iniciais, para cada data set em particular, que maximizaria a acurácia preditiva das regras de classificação descobertas.

Referências Bibliográficas

WITTEN, Ian H.; FRANK, Eibe. **Data Mining: Practical Machine Learning Tools and Techniques**. 2. Ed. San Francisco: Morgan Kaufmann, 2005.

PARPINELLI, R. S.; LOPES H. S.; FREITAS, A. A. **Data mining with an ant colony optimization algorithm**. IEEE Transactions on Evolucionary Computing 6(4), p. 321-332, 2002a.

PARPINELLI, R. S.; LOPES H. S.; FREITAS, A. A. **An Ant Colony Algorithm for Classification Rule Discovery**. Data Mining: a Heuristic Approach, Idea Group Publishing, p. 191-208, 2002b.

PARPINELLI, R. S.; LOPES H. S.; FREITAS, A. A. **Ant colony algorithms for data classification**. Khosrow-Pour, M (Ed.), Encyclopedia of Information Science and Technology**,** 2nd Edition. Hershey: IGI Publishing, p. 420-424, 2008.

NAISBITT, John; ABURDENE Patricia. **Megatrends 2000: New Directions for Tomorrow**. New York: Avon Books, 1990.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMITH, P.; UTHURUSAMY, R**. Advances in Knowledge Discovery and Data Mining**. Menlo Park: AAAI/MIT Press, 1996.

AMO, Sandra A. **Introdução a Mineração de Dados**. Universidade Federal de Uberlândia, p. 87-97, 2003.

DENEUBOURG, J. L.; ARON, S.; GOSS, S.; PASTEELS, J. M. **The self-organizing exploratory pattern of the Argentine ant.** Journal of Insect Behavior, vol. 3, p. 159, 1990.

GOSS, S.; ARON, S.; DENEUBOURG, J. L.; PASTEELS, J. M. **Self-organized shortcuts in the Argentine ant.** Naturwissenschaften, vol. 76, p. 579–581, 1989.

DORIGO, M; BIRATTARI, M; STÜTZLE, T. **Ant Colony Optimization – Artificial Ants as a Computational Intelligence Technique**. IEEE Computational Intelligence Magazine, p. 28-39, 2006.

DORIGO, M.; DI CARO, G. **The Ant Colony Optimization meta-heuristic.** New Ideas in Optimization. McGraw Hill, Londres, p. 11–32, 1999.

DORIGO, M.; DI CARO, G.; GAMBARDELLA, L.M. **Ant algorithms for discrete optimization**. Artificial Life, vol. 5, no. 2, p. 137–172, 1999.

MAMDANI, E. H. **Advances in the linguistic syntesis of fuzzy controllers**. International Journal of Man-Machine Studies, p. 669 – 678, 1976.

MAMDANI, E. H; PROCYK, T.; BAAKLINI, N. **Application of Fuzzy Logic to Controller Design Based on Linguistic Protocol**. Discrete Systems and Fuzzy Reasoning, Queen Mary College, Universidade de Londres, 1979.

SANDRI, S.; CORREA, C. **Lógica Nebulosa**. Conselho Nacional de Redes Neurais, INPE, p. c073-c090, 1999.

ROZIN, V.; MARGALIOT, M. **The Fuzzy Ant**. IEEE Computational Intelligence Magazine, p.18-28, 2007.

VIOT, G. **Fuzzy Logic in C – Creating a fuzzy-based inference engine**. Dr. Dobb's journal, p. 40-49, 1993.

CRUZ, A. J. O. **Lógica Nebulosa**. Universidade Federal do Rio de Janeiro, 2004.

FRANK, A.; ASUNCION, A. **UCI Machine Learning Repository.** Irvine, CA: University of California, School of Information and Computer Science, 2010. Disponível em <http://archive.ics.uci.edu/ml>. Acesso em junho, 2011.