GUI Ant-Miner: Uma versão atualizada do minerador de dados baseado em colônias de formigas

Fernando Meyer[1], Rafael Stubs Parpinelli[1]

[1]Departamento de Ciência da Computação – Universidade do Estado de Santa Catarina (UDESC – Joinville)
Campus Universitário – CEP 89228-100 – Joinville – SC – Brasil
fernando.meyer@gmail.com, parpinelli@joinville.udesc.br

Abstract. *This paper presents an updated version of the system called Ant-Miner (Ant Colony-based Data Miner). The GUI Ant-Miner (Graphical User Interfaced Ant-Miner) presents all the features from the original algorithm with some improvements: friendly graphical interface for user/system interactions; make capable the use of ants population concept; data input file standardized with the well know Weka system. Comparing the obtained results both versions reached similar solutions in the data set utilized.*

Resumo. *Este artigo apresenta uma versão atualizada do sistema chamado Ant-Miner (Minerador de Dados Baseado em Colônias de Formigas). O GUI Ant-Miner (Graphical User Interfaced Ant-Miner) apresenta todas as características do algoritmo original com algumas melhorias: interface gráfica amigável para interações entre usuário e sistema; possibilita o uso do conceito de população de formigas; arquivo de entrada dos dados padronizado com o sistema Weka. Comparando os resultados obtidos, ambas versões encontraram soluções semelhantes na base de dados utilizada.*

Palavras-chave. Mineração de Dados, Regras de Classificação, Otimização por Colônias de Formigas.

1. Introdução

O computador vem se tornando uma ferramenta de suma importância para o auxílio à tomada de decisões. Empresas armazenam enormes quantidades de dados de clientes, produtos, mercados, processos de manufatura, tornando a análise e a interpretação manuais desses dados um processo difícil e demorado [Han e Kamber 2000]. Com o uso de técnicas de mineração de dados é possível gerar informações para auxiliar na análise e descoberta de conhecimento.

Uma forma freqüente de representar o conhecimento descoberto são regras de previsão (ou classificação) do tipo [Parpinelli, Lopes e Freitas 2002]:

SE <condições> ENTÃO <classe>

A parte <*condições*> é o antecedente da regra e pode ser constituída por uma ou mais condições. A parte <*classe*> é o conseqüente da regra que satisfaz a parte antecedente.

Em Parpinelli, Lopes e Freitas (2002) foi proposto um algoritmo para a descoberta de regras de classificação chamado *Ant-Miner* (*Ant Colony-based Data Miner* – Minerador de Dados Baseado em Colônias de Formigas).

A heurística utilizada pelo *Ant-Miner*, Otimização por Colônias de Formigas (*Ant Colony Optimization* – ACO), foi proposta por Dorigo, Colorni e Maniezzo (1996) como uma nova heurística para resolver problemas de otimização combinatorial. A heurística ACO é inspirada na observância de que as formigas são capazes de encontrar o menor caminho entre o ninho e uma fonte de alimento.

No *Ant-Miner*, a escolha de um caminho por uma formiga equivale à escolha de uma condição para ser adicionada na regra parcial construída pela formiga (agente do sistema). Cada condição é constituída por uma tripla: nome de um atributo, o operador "=" e o respectivo valor, tal como <*Gênero = feminino*>.

O *GUI Ant-Miner* é uma versão atualizada, cujas contribuições são: permitir usar o conceito de populações inserido na heurística ACO; padronização do arquivo de entrada de dados e utilização de uma interface gráfica.

O texto está organizado da seguinte forma: no Capítulo 2 está descrita a inspiração para a heurística ACO; no Capítulo 3 está descrita a versão original do algoritmo *Ant-Miner*; a versão *GUI Ant-Miner* está detalhada no Capítulo 4; no Capítulo 5 estão descritos os resultados obtidos; e no Capítulo 6 estão as conclusões e trabalhos futuros.

2. Otimização por Colônias de Formigas

A heurística ACO é inspirada na seguinte observância da natureza: capazes de encontrar o menor caminho entre uma fonte de alimento e o ninho, as formigas utilizam uma substância chamada feromônio como forma de comunicação entre si. Ao se deslocarem, as formigas formam uma trilha dessa substância, que se torna mais atraente na medida em que mais formigas escolhem a mesma trilha e depositam mais feromônio. Individualmente uma formiga pode não produzir boas soluções para o problema em questão, mas através da cooperação são capazes de encontrar boas soluções [Dorigo e Gambardella 1997]. Este comportamento pode ser visto na Figura 1.

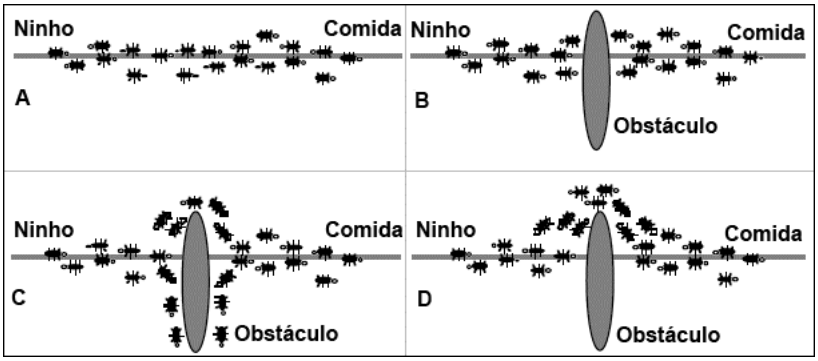

Figura 1. Comportamento das formigas após o surgimento de um obstáculo entre o ninho e uma fonte de alimento [Dorigo e Gambardella 1997]

Através desta observância, a metodologia ACO é aplicada como uma nova heurística para resolver problemas de otimização combinatorial.

3. O Algoritmo *Ant-Miner*

A ferramenta *Ant-Miner* foi desenvolvida utilizando a linguagem C e interage com o usuário através de uma interface baseada em texto. Para carregar as bases de dados, o usuário deve utilizar arquivos de texto com formato específico para a ferramenta.

Uma descrição de alto nível do *Ant-Miner* é mostrada na Figura 2. O algoritmo segue uma seqüência de passos para descobrir uma lista ordenada de regras de classificação que cubra todos ou quase todos os casos de treinamento. No início do processo de descoberta das regras a lista que armazena as regras é uma lista vazia e o conjunto de treinamento é composto por todos os casos de treinamento. No inicio do laço *ENQUANTO*, todas as trilhas (estados do problema) são inicializadas com a mesma quantidade de feromônio. Cada iteração do laço *ENQUANTO*, correspondente ao número de execuções do laço *REPITA-ATÉ*, descobre uma regra de classificação. Esta regra é adicionada à lista de regras descobertas e os casos de treinamento que são corretamente cobertos por esta regra (i.e. os casos que satisfazem o antecedente da regra e têm a classe prevista pelo seu conseqüente) são removidos do conjunto de treinamento. Este processo é iterativamente executado enquanto o número de casos de treinamento for maior do que um parâmetro especificado pelo usuário, chamado *Max_casos_n_cobertos* (casos não cobertos no conjunto de treinamento).

```
ConjuntoTreinamento = {todos os casos de treinamento };
ListaDeRegras = []; /* lista das regras é inicializada como lista vazia */
ENQUANTO (ConjuntoTreinamento > Max_casos_n_cobertos)
    i = 1; /* índice de cada formiga */
    j = 1; /* índice do teste de convergência */
    Inicialize todas trilhas com a mesma quantidade de feromônio;
    REPITA
        Ant_i começa com uma regra vazia e incrementalmente constrói uma regra de classificação R_i,
        adicionando um termo por vez à regra atual;
        Pode a regra R_i;
        Atualize o feromônio de todas as trilhas, incrementando o feromônio na trilha seguida pela formiga
        Ant_i (proporcionalmente a qualidade da regra R_i) e decrementando o feromônio nas outras trilhas
        (simulando a evaporação do feromônio);
        SE (R_i for igual a R_i – 1) /* atualiza teste de convergência */
            ENTÃO j = j + 1;
            SENÃO j = 1;
        FIM-SE
        i = i + 1;
    ATÉ (i ≥ Num_formigas) OU (j ≥ Num_regras_converg)
    Escolha a melhor regra R_m dentre todas as regras R_i construídas pelas formigas;
    Adicione a regra R_m na ListaDeRegras;
    ConjuntoTreinamento = ConjuntoTreinamento - {grupo de casos corretamente cobertos por R_m};
FIM-ENQUANTO
Adicione a regra padrão na última posição de ListaDeRegras.
```

Figura 2. Uma descrição do algoritmo *Ant-Miner* [Parpinelli, Lopes e Freitas 2002]

4. O *GUI Ant-Miner*

O *GUI Ant-Miner* baseia-se no algoritmo *Ant-Miner* mas possui três diferenças principais: permite utilizar o conceito de populações em otimização por colônias de formigas, padroniza o arquivo de entrada de dados e utiliza uma interface gráfica. Além disso, o *GUI Ant-Miner* foi desenvolvido usando-se a linguagem *Java*.

No *GUI Ant-Miner* utilizam-se populações de formigas para a otimização com o objetivo de melhorar a pesquisa no espaço de busca. Ou seja, apenas a melhor regra, dentre um conjunto de *n* regras construídas por *n* formigas de cada população, causa alteração nas trilhas de feromônio. No *Ant-Miner*, cada regra construída por uma formiga resulta na atualização das trilhas de feromônio. Para executar o *GUI Ant-Miner* com as características da versão original, basta ajustar o número de formigas igual a 1. A Figura 3 descreve o algoritmo implementado no *GUI Ant-Miner*, onde, em negrito, estão as operações que diferem da versão original.

```
ConjuntoTreinamento = {todos os casos de treinamento };
ListaDeRegras = []; /* lista das regras é inicializada como lista vazia */
ENQUANTO (ConjuntoTreinamento > Max_casos_n_cobertos)
    j = 1; /* índice do teste de convergência */
    Inicialize todas trilhas com a mesma quantidade de feromônio;
    ENQUANTO (iteração ≤ númeroDeIterações) OU (j ≥ Num_regras_converg)
        i = 1; /* índice de cada formiga */
        REPITA
            Ant_i começa com uma regra vazia e incrementalmente constrói uma regra de classificação R_i,
            adicionando um termo por vez à regra atual;
            Pode a regra R_i;
            i = i + 1;
        ATÉ (i ≥ Num_formigas)
        Dentre as regras R_i construídas pela população de formigas, escolha a melhor regra R_m
        construída pela formiga Ant_m;
        Atualize o feromônio de todas as trilhas, incrementando o feromônio na trilha seguida pela melhor
        formiga Ant_m (proporcionalmente à qualidade da regra R_m) e decrementando o feromônio nas outras
        trilhas (simulando a evaporação do feromônio);
        SE (R_m na iteração corrente for igual a R_m na iteração anterior) /* atualiza teste de convergência */
            ENTÃO j = j + 1;
            SENÃO j = 1;
        FIM-SE
        iteração = iteração + 1;
    FIM ENQUANTO
    Adicione a regra R_m na ListaDeRegras;
    ConjuntoTreinamento = ConjuntoTreinamento - {grupo de casos corretamente cobertos por R_m};
FIM-ENQUANTO
Adicione a regra padrão na última posição de ListaDeRegras;
```

Figura 3. Uma descrição do algoritmo no *GUI Ant-Miner*

Verifica-se na Figura 3 a inclusão de um laço *ENQUANTO* que executa um número de iterações especificado pelo usuário, onde cada iteração representa uma geração da colônia de formigas, ou até que as regras construídas tenham convergido, ou seja, durante um número específico de gerações, *Num_regras_converg*, a colônia de formigas percorreu o mesmo caminho (construiu a mesma regra). Verifica-se também

que a escolha da melhor regra é feita dentre as regras construídas pela população, lembrando que, de cada geração, extrai-se apenas uma regra.

Outra contribuição do *GUI Ant-Miner* é a padronização do arquivo de entrada de dados. O *Ant-Miner* utiliza um formato próprio, enquanto o *GUI Ant-Miner* usa o formato ARFF (*Attribute-Relation File Format*). Este formato é o mesmo usado pelo sistema Weka[1], desenvolvido na universidade de Waikato, na Nova Zelândia.

A Figura 4 mostra a tela inicial do *GUI Ant-Miner* após a abertura de uma base de dados. No *menu*, indicado pela seta 1, existem as opções: *File–Open*, *File–Exit* e *Help–About GUI Ant-Miner*. A opção *File–Open* permite abrir uma base de dados; *File–Exit* fecha a aplicação; e *Help–About GUI Ant-Miner* mostra os responsáveis pelo desenvolvimento da ferramenta. A seta 2 indica as abas de opções: *Preprocess* e *Classify*.

Na aba *Preprocess*, a área *Relation*, indicada pela seta 3 da Figura 4, mostra o título da base (*Relation*), o número de casos (*Instances*) e o número de atributos (*Attributes*). A área *Attributes*, indicada pela seta 4, mostra todos os atributos da base. A classe (ou atributo-meta) é sempre o último atributo mostrado nesta área. A área *Selected Attribute*, indicada pela seta 5, mostra informações sobre o atributo selecionado na área *Attributes*: o nome do atributo (*Name*), quantidade de valores ausentes desse atributo nos casos e a respectiva porcentagem (*Missing*), e o número de valores distintos que o atributo pode assumir (*Distinct*). Ainda nesta área são mostrados os valores que o atributo pode assumir e o número de vezes que este valor ocorre na base de dados (seta 6). A área indicada pela seta 7 está reservada para a visualização de gráficos relativos à distribuição dos atributos na base de dados. Os gráficos não estão implementados nessa versão da interface.

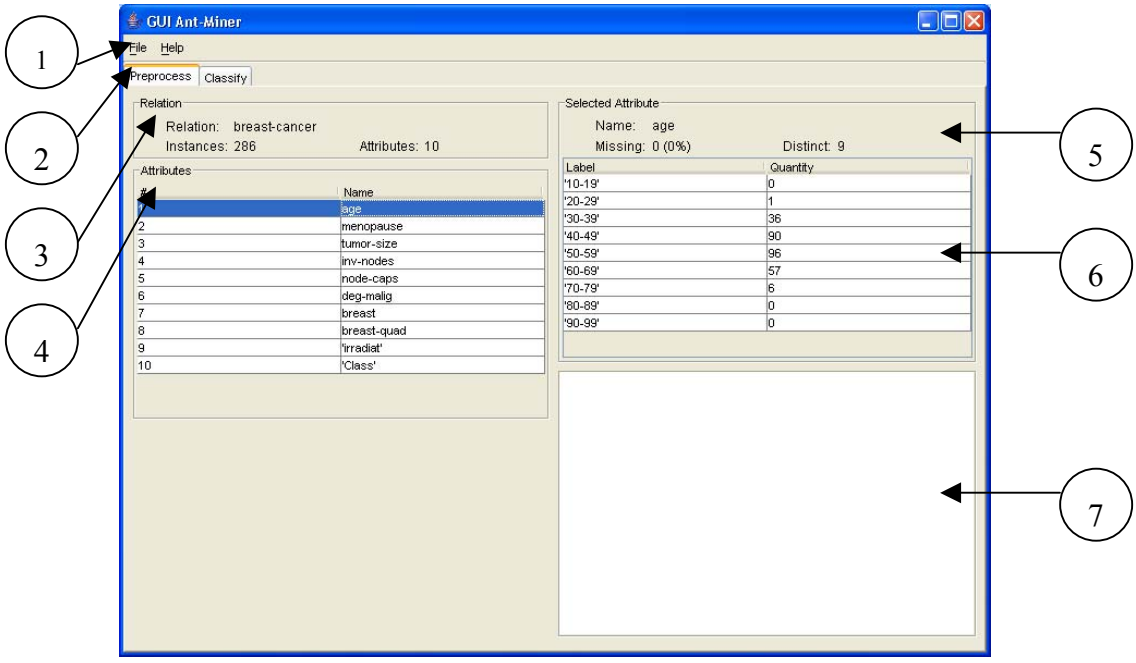

Figura 4. Tela inicial do *GUI Ant-Miner*

[1] http://www.cs.waikato.ac.nz/~ml/

Na aba *Classify* o usuário pode alterar parâmetros do algoritmo, iniciar a tarefa de classificação e visualizar os resultados da tarefa. Esta aba só é habilitada após o usuário abrir uma base de dados válida, no formato ARFF. Selecionando-se a aba *Classify*, é apresentada ao usuário a tela demonstrada na Figura 5. Nesta tela o usuário informa os parâmetros do algoritmo através dos campos indicados pelas setas 1 e 2, altera opções de impressão dos resultados, indicados pela seta 3, inicia a tarefa de classificação clicando no botão *Start* indicado pela seta 4, e visualiza os resultados na área indicada pela seta 5.

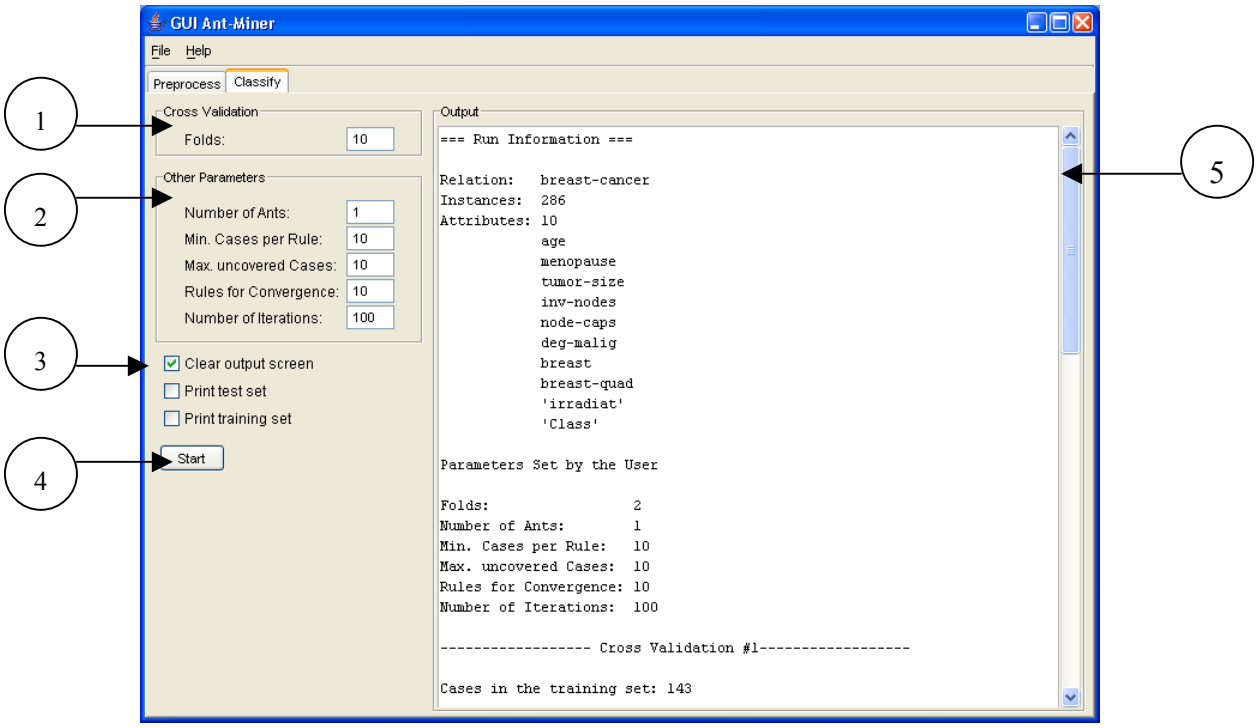

Figura 5. A aba *Classify* no *GUI Ant-Miner*

5. Resultados

Para a comparação dos resultados obtidos com o *GUI Ant-Miner* e o *Ant-Miner*, usou-se uma base de dados de domínio público disponível no repositório da Universidade da Califórnia (UCI – *University of California at Irvine*) [Hettich, Blake e Merz 1998]. Trata-se da base *Ljubljana Breast Cancer*.

Para uma comparação justa, os parâmetros usados foram os mesmos nos dois sistemas: número mínimo de casos cobertos por regra *Min_casos_por_regra* = 10; número máximo de casos não cobertos na base de treinamento *Max_casos_n_cobertos* = 10; número de regras para teste de convergência *Num_regras_converg* = 10; Número de partições para a validação cruzada = 10. No *GUI Ant-Miner* ainda se definiu valor 100 para *Número de Iterações* e valor 1 para *Número de Formigas*.

Os resultados são mostrados na Tabela 1. Os valores que estão à esquerda do símbolo "±" são as médias dos resultados das validações cruzadas e os valores que estão à direta são os desvios-padrão dos resultados correspondentes.

Tabela 1: Resultados obtidos na base *Ljubljana Breast Cancer*

	Taxa de acerto	No. de regras	No. de condições	Tempo (segundos)
GUI Ant-Miner	**74,84% ± 2,00%**	**5,80 ± 0,13**	**7,40 ± 0,27**	**48**
Ant-Miner	**75,28% ± 2,24%**	**7,10 ± 0,31**	**9,10 ± 0,38**	**6**

Em uma primeira análise dos resultados verifica-se que estes são diferentes e isto se justifica pela característica probabilística inerente aos sistemas. Em uma análise um pouco mais detalhada verifica-se que os resultados obtidos para a taxa de acerto nos dois sistemas são tecnicamente iguais considerando-se os valores de desvio-padrão. Ou seja, somando-se a taxa de acerto do *GUI Ant-Miner* ao seu desvio-padrão, atinge-se o resultado obtido pelo *Ant-Miner*. O mesmo pode ser dito sobre a taxa de acerto do *Ant-Miner* em relação a do *GUI Ant-Miner*.

A respeito do número de regras e do número de condições, o *GUI Ant-Miner* apresentou resultados melhores que o *Ant-Miner*. No entanto, a diferença observada é pequena e condiz com um resultado esperado, considerando que os algoritmos em comparação aqui são fundamentalmente os mesmos.

O tempo de processamento necessário para gerar as regras foi um quesito no qual o *GUI Ant-Miner* se mostrou inferior ao *Ant-Miner*. O primeiro foi 8 vezes mais lento, isto em função da linguagem de programação *Java* utilizada, como também de algumas lógicas e estruturas de dados utilizadas.

6. Conclusões e Trabalhos Futuros

Neste trabalho foi apresentado uma versão atualizada do algoritmo *Ant-Miner*, chamado aqui de *GUI Ant-Miner*. O *GUI Ant-Miner* (*Graphical User Interfaced Ant-Miner*) foi implementado com todas as características do algoritmo original possuindo algumas melhorias: interface gráfica amigável; possibilidade de utilizar o conceito de população de formigas; arquivo de entrada dos dados padronizado com o sistema Weka.

A avaliação geral do *GUI Ant-Miner*, dada a análise dos resultados, foi bastante positiva, atingindo os resultados esperados.

Algumas direções para trabalhos futuros são:

- Verificar a influência do número de formigas em bases de dados de grande porte onde o espaço de busca é extenso;
- Melhorar o *feedback* da ferramenta ao usuário enquanto se executa a tarefa de classificação, mostrando uma barra de progresso da tarefa e possibilitando ao usuário cancelar a operação;
- Implementar a visualização de gráficos para que o usuário possa ter uma melhor idéia da distribuição dos atributos na base de dados;
- Melhorar o desempenho da ferramenta através da otimização de algumas lógicas envolvidas no algoritmo e do experimento de diferentes estruturas de dados disponíveis na linguagem *Java*;

- Possibilitar ao usuário conectar a bases de dados através de *JDBC* (uma biblioteca para conexão a bases de dados) como alternativa ao formato texto usado atualmente.

Referências

Dorigo, M., Colorni, A., Maniezzo, V. (1996) "The Ant System: Optimization by a Colony of Cooperating Agents", Proceedings of the IEEE Transactions on Systems, Man, and Cybernetics-Part B, p. 29-41.

Dorigo M., Gambardella L. M. (1997) "Ant Colonies for the Traveling Salesman Problem", BioSystems, v. 43, p. 73-81.

Han, J., Kamber, M., Data Mining: Concepts and Techniques, Morgan Kaufmann Series in Data Management Systems, 2000.

Hettich, S., Blake, C. L., Merz C. J. (1998) "UCI Repository of Machine Learning Databases", http://www.ics.uci.edu/~mlearn/MLRepository.html, Department of Information and Computer Science, University of California at Irvine, July.

Parpinelli, R.S., Lopes, H.S., Freitas, A.A. (2002) "Data mining with an ant colony optimization algorithm", Proceedings of the IEEE Transactions on Evolutionary Computation, special issue on Ant Colony Algorithms, August, v. 6, n. 4, p. 321-332.