Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Colegiado de Sistemas de Informação

Aplicação da Mineração de Dados para a recomendação de parâmetros para o COIN-OR *Branch and Cut*

Rafael de Sousa Oliveira Martins

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Janniele Aparecida Soares Araujo

Agosto, 2016
João Monlevade/MG

M386a Martins, Rafael de Sousa Oliveira.
Aplicação da mineração de dados para a recomendação de parâmetros para o COIN-OR Branch and Cut [manuscrito] / Rafael de Sousa Oliveira Martins. - 2016.

56f.: il.: color; tabs.

Orientador: Prof. Me. Janniele Aparecida Soares Araújo.
Coorientador: Prof. Dr. Samuel Souza Brito.

Monografia (Graduação). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Departamento de Computação e Sistemas de Informação.

2. Otimização . 3. Mineração de dados (Computação). 4. Descoberta de Conhecimento em Base de Dados. I. Araújo, Janniele Aparecida Soares. II. Brito, Samuel Souza. III. Universidade Federal de Ouro Preto. IV. Titulo.

Catalogação: ficha@sisbin.ufop.br CDU: 004.6/.65

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E APLICADAS
COLEGIADO DO CURSO DE SISTEMAS DE INFORMAÇÃO

TERMO DE RESPONSABILIDADE

Eu, Rafael de Sousa Oliveira Martins, declaro que o texto do trabalho de conclusão de curso intitulado *"Aplicação da Mineração de dados para Recomendação de Parâmetros para o COIN-OR"* é de minha inteira responsabilidade e que não há utilização de texto, material fotográfico, código fonte de programa ou qualquer outro material pertencente a terceiros sem as devidas referências ou consentimento dos respectivos autores.

João Monlevade, 10, de Agosto de 2016.

Rafael de Sousa Oliveira Martins

Rua Trinta e seis, 115 – Bairro Loanda – CEP 35931-008 – João Monlevade – MG – Brasil
http://www.icea.ufop.br – cosi@decea.ufop.br – **(31) 3852-8709**

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E APLICADAS
COLEGIADO DO CURSO DE SISTEMAS DE INFORMAÇÃO

ATA DE DEFESA

Aos dez dias do mês de agosto de dois mil e dezesseis, às dezenove horas e zero minutos, na sala C304 do Instituto de Ciências Exatas e Aplicadas, foi realizada a defesa de Monografia pelo aluno **Rafael de Sousa Oliveira Martins,** sendo a Comissão Examinadora constituída pelos professores: Prof. Msc. Janniele Aparecida Soares Araújo, Prof. Msc. Samuel Souza Brito, Prof. Msc. Helen de Cássia Sousa da Costa Lima e Prof. Msc. Matheus Guedes Vilas Boas.

O candidato apresentou a monografia intitulada: *"Aplicação da Mineração de dados para a Recomendação de Parâmetros para o COIN-OR".* A comissão examinadora deliberou, por unanimidade, pela aprovação do candidato, com nota 9,5 (nove e meio), concedendo-lhe o prazo de 15 dias para incorporação das alterações sugeridas ao texto final. Na forma regulamentar, foi lavrada a presente ata que é assinada pelos membros da Comissão Examinadora e pelo graduando.

João Monlevade, 10 de Agosto de 2016.

Janniele Aparecida Soares Araujo
Prof. Msc. Janniele Aparecida Soares Araújo
Professor Orientador/Presidente

Samuel Souza Brito
Prof. Msc. Samuel Souza Brito
Professor Coorientador

Helen de C. S. Costa Lima
Prof. Msc. Helen de Cássia Sousa da Costa Lima
Professor Convidado

Matheus Guedes Vilas Boas
Prof. Msc. Matheus Guedes Vilas Boas
Professor Convidado

Rafael de Sousa Oliveira Martins
Rafael de Sousa Oliveira Martins
Graduando

Dedico este trabalho para todos aqueles que fizeram do meu sonho real, me proporcionando forças para que eu não desistisse de ir atrás do que eu buscava para minha vida. Muitos obstáculos foram impostos para mim durante esses últimos anos, mas graças a vocês eu não fraquejei. Obrigado por tudo Pai, Mãe, Thais, Prof. Msc. Janniele, Professores, amigos e colegas.

Agradecimentos

Em primeiro lugar agradeço aos meus pais, pela confiança e investimento durante a realização deste objetivo.

Agradeço a minha namorada Thais, que mesmo um longo período distante, conseguiu me motivar, apoiar e quando precisava dava uns puxões de orelha, além de todo seu carinho que sem ele não seria capaz a realização desde trabalho.

Agradeço a todos os professores aos quais tive o prazer de ser aluno, em especial a Prof[a]. Msc. Janniele, pela excelente orientação, pelo suporte, disponibilidade, comprometimento e o compartilhamento de conhecimento.

Agradeço ao Coorientador Prof. Msc. Samuel que apesar do pouco tempo, me ajudou muito.

Agradeço aos meus amigos, em especial aos indivíduos que dividiram não somente a casa, mas as felicidades e as tristezas, em especial o Rhivison, Guilherme, Douglas, Edgar, Igor, Giovanne e Matheus, obrigado mesmo, vocês são "fodas".

Agradeço a Riot, pelos momentos de descontração que me ajudaram a seguir diante das adversidades da vida.

Agradeço a Universidade Federal de Ouro Preto, pelo apoio financeiro que me ajudou na permanência na universidade.

E por fim, agradeço a Deus que sem Ele não existiria nada, e nada seria possível.

O aprendizado adquirido aqui, e o prazer imenso de desenvolver esse trabalho serão únicos. Obrigado pela oportunidade, foi um prazer tê-los comigo.

"É preciso força pra sonhar e perceber que a estrada vai além do que se vê".
(Los Hermanos)

Resumo

Este trabalho propõe a recomendação de parâmetros para o resolvedor *COIN-OR Branch and Cut (CBC)*, com o intuito de obter a melhor configuração para problemas de otimização e melhorar a eficiência na busca da solução ótima. Tal situação pode ser resolvida por meio da aplicação do processo de Descoberta de Conhecimento em Base de Dados (*Knowledge Discovery in Databases - KDD*). Os resultados foram satisfatórios, alcançados por meio do desenvolvimento de uma aplicação para classificar novas configurações de parâmetros para problemas de otimização. Além disso, foi possível encontrar um modelo baseado em árvore, genérico e confiável, para diferentes tipos de problemas.

Palavras-chaves: COIN-OR Branch and Cut. Otimização Combinatória. Recomendação de Parâmetros. Descoberta de conhecimento em Base de Dados.

Abstract

This work proposes a parameter recommendation for the *COIN-OR Branch and Cut (CBC)* solver in order to achieve the best configuration for optimization problems and to improve the efficiency in the pursuit of an optimal solution. This situation can be solved through the use of the Knowledge Discovery Process Application Data Base (*Knowledge Discovery in Databases - KDD*). The results obtained were satisfactory, obtained by the development of an application to classify new configurations of parameters for optimization problems. Moreover, it was possible to find a generic and reliable tree-based model for different kinds of problems.

Key-words: COIN-OR Branch and Cut. Combinatorial Optmization. Parameter Recommendations. Knowledge Discovery in Databases.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

UFOP	Universidade Federal de Ouro Preto
KDD	Knowledge Discovery in Databases
WEKA	Waikato Environment for Knowledge Analysis
GOAL	Group of Optimization and Algorithms
COIN-OR	COmputational INfrastructure for Operations Research
PI	Programação Inteira
CC	Correlation Coefficient
MAE	Mean Absolute Error
RMSE	Root Mean Squared Error
RAE	Root Absolut Error
RRSE	Root Relative Squared Error
CCI	Correctly Classified Instances
DASA	Differential Ant-Stigmergy Algotithm
PSO	Particle Swarm Optimization
.arff	Modelos padrão de entrada de dados
.csv	Modelos de arquivo separado por vírgulas

Sumário

1 Introdução

"Vivemos na era da informação" é um ditado popular, no entanto a verdade é que estamos vivendo na era dos dados (HAN, 2005). De acordo com Han (2005), existe um grande crescimento de dados coletados da sociedade, empresas, ciência e engenharia, medicina e outros aspectos da vida diária. Com essa crescente na quantidade de dados coletados, surgem naturalmente questões como: "O que fazer com todos estes dados armazenados?", "Como analisar e utilizar de maneira útil todos estes dados?", entre diversas outras. A análise de grandes quantidade de dados se torna então, uma tarefa desafiadora para humanidade. Essa tarefa torna imprescindível a utilização e desenvolvimento de ferramentas computacionais, que, de forma automática e inteligente na tarefa de analisar, interpretar e relacionar os dados, desenvolve e seleciona uma estratégia em cada contexto de aplicação. No presente trabalho, será desenvolvida uma ferramenta para a recomendação de parâmetros para o COIN-OR, um resolvedor para problemas de Programação Linear Inteira.

A Programação Linear Inteira, também referida como Programação Inteira (PI), pode ser vista como um problema de Programação Matemática em que a função objetivo, bem como as restrições, são lineares, porém uma ou mais variáveis de decisão podem assumir apenas valores inteiros. Uma forma de solução de problemas de PI se dá por meio da utilização do resolvedor de código aberto *COIN-OR Branch and Cut (CBC)*[1] (LOUGEE-HEIMER, 2003).

O *CBC* possui uma série diversificada de parâmetros, que podem influenciar diretamente no desempenho da resolução de um problema. Dessa forma, a tarefa de identificar e utilizar os melhores parâmetros para um dado problema é complexa, porém essencial. Tal situação pode ser resolvida por meio da aplicação do processo de Descoberta de Conhecimento em Base de Dados (*Knowledge-Discovery in Databases - KDD*), que segundo Fayyad, Piatetsky-Shapiro e Smyth (1996) é o processo que visa o descobrimento de conhecimento válido, relevante e desconhecido em base de dados. Esse processo é dividido em 5 etapas: seleção, pré-processamento, transformação, mineração de dados e interpretação.

A fim de resolver o problema e atender os objetivos do presente trabalho, faz-se necessário o uso da mineração de dados, que é definida como um processo de retirada de informações implícitas de maneira não trivial. Tal etapa é uma parte integrante do processo de *Knowledge-Discovery in Databases (KDD)*. As tarefas utilizadas no trabalho foram: classificação e regressão. O objetivo dessas tarefas é prever o estado mais provável

[1] <https://projects.coin-or.org/Cbc>

que uma variável pode atingir a partir das características de um conjunto de registros históricos, diferenciando apenas em como cada uma das tarefas executam o objetivo.

A base de dados empregada neste trabalho foi um conjunto de problemas de Programação Linear Inteira resolvidos pelo *COIN-OR Branch and Cut (CBC)*, utilizadas no Grupo de Otimização e Algoritmos (GOAL/UFOP) da Universidade Federal de Ouro Preto. Tal grupo desenvolve pesquisas nas áreas de otimização mono e multi-objetivo, inteligência computacional, mineração de dados, tratamento e recuperação de informação e algoritmos. O grupo é formado por professores envolvidos com a indústria, bem como alunos de pós-graduação e graduação. Esses problemas envolvem uma variedade de problemas do mundo real, tais como alocação de horários, escalonamento de projetos, roteamento de veículos, logística reversa, arte, medicina, engenharia e economia.

O *COIN-OR Branch and Cut (CBC)* é um resolvedor *open-source* de Programação Inteira Mista escrito em C++, que possui uma diversidade de parâmetros. Os parâmetros que são utilizados nos problemas desenvolvidos na GOAL/UFOP são utilizados de forma empírica.

Dessa forma, o presente trabalho é apresentado a fim de se utilizar o processo de mineração de dados com a tarefa de classificação e regressão para recomendar quais parâmetros utilizar, de maneira fidedigna, dado um novo problema a ser executado no CBC.

1.1 Objetivos

Este capítulo constituído por dois sub tópicos, o primeiro apresentando os objetivos de forma generalizada, e outro, a partir da generalização dos objetivos explica de maneira mais específica cada um do objetivos.

1.1.1 Objetivo Geral

O objetivo geral do presente trabalho é a recomendação, de maneira confiável, de quais parâmetros utilizar para um novo problema a ser executado no *(*COIN-OR Branch and Cut). Especificamente, a recomendação trata dos dados extraídos da base dados do GOAL/UFOP, aplicando modificações de pré-processamento e identificando quais as influências de cada um destes dados no conjunto de instâncias. A fim de efetuar uma análise sobre os fatores que têm influência no resultado da execução do problema no *CBC*, são utilizadas as tarefas de classificação e regressão. Essas tarefas são capazes de executar algoritmos que retornem informações relevantes e que, até momento, não são conhecidas. Além disso, serão comparados os algoritmos das tarefas de classificação e regressão para o problema em questão, a fim de conseguir identificar aqueles que retornem resultados importantes a termos de informações. Desse modo, espera-se a obtenção de parâmetros

confiáveis, que permitam, além de analisar os resultados obtidos, apresentá-los de maneira intuitiva, detalhando os índices de confiabilidade e a melhoria provida com a resolução do problema em questão.

1.1.2 Objetivos Específicos

Buscando atingir o objetivo geral mencionado anteriormente, têm-se a necessidade de atingir os objetivos específicos listados a seguir:

- analisar e preprocessar os dados da base de dados;
- efetuar modificações específicas na base de dados para a tarefa da classificação e de regressão;
- identificar a influência dos atributos de cada registro da base de dados;
- selecionar os algoritmos classificadores e de predição.
- comparar os algoritmos selecionados de acordo com a sua tarefa;
- identificar os melhores algoritmos, que tenham resultados relevantes e confiáveis para cada tarefa;
- implementar uma biblioteca contendo os melhores algoritmos;
- apresentar os resultados de maneira intuitiva para o usuário;
- analisar os resultados, detalhando os índices de confiabilidade e a melhoria provida com a resolução do problema em questão.

2 Revisão Bibliográfica

2.1 Descoberta de Conhecimento em Base de Dados

O termo *Knowledge Discovery in Databases (KDD)*, que no português pode ser descoberta de conhecimento em base de dados, foi formalizado em 1989 em referência ao conceito de busca por conhecimento a partir de base de dados. Uma das definições mais populares foi proposta por Fayyad, Piatetsky-Shapiro e Smyth (1996): "*KDD* é um processo de várias etapas, não trivial, interativo e iterativo, para identificação de padrões compreensíveis, válidos, novos e potencialmente úteis a partir de grandes conjuntos de dados".

O *KDD* é o processo que têm como objetivo a procura de conhecimento potencialmente útil em base dados, sendo composto por, basicamente, 7 passos. De acordo com Han (2005) os passos são estabelecidos como descrito abaixo e mostrado na Figura 1:

1. **Limpeza dos dados:** remover o ruído e os dados inconsistentes.
2. **Integração dos dados:** combinar múltiplas fontes de dados.
3. **Seleção dos dados:** selecionar os dados relevantes para a análise.
4. **Transformação dos dados:** transformar e consolidar os dados de forma adequada para mineração de dados, através da realização de operações de agregação e sumarização.
5. **Mineração de dados:** processo essencial onde são aplicados métodos inteligentes para extrair padrões de dados.
6. **Avaliação de padrões:** identificar os padrões verdadeiramente interessantes que representam o conhecimento.
7. **Apresentação do conhecimento:** utilização de técnicas de visualização e representação para representação do conhecimento extraído para os usuários.

As etapas de 1 a 4 são formas diferentes de preprocessamento de dados, onde os dados são preparados para a mineração. A mineração de dados é o processo de descobrir padrões e conhecimentos relevantes de grande quantidade de dados. As fontes de dados podem ser banco de dados, *data warehouses*, repositórios de informação na *Web* ou dados que são transmitidos de forma dinâmica (HAN, 2005). A base de dados a ser trabalhada neste presente trabalho encontra-se disponível em um banco de dados.

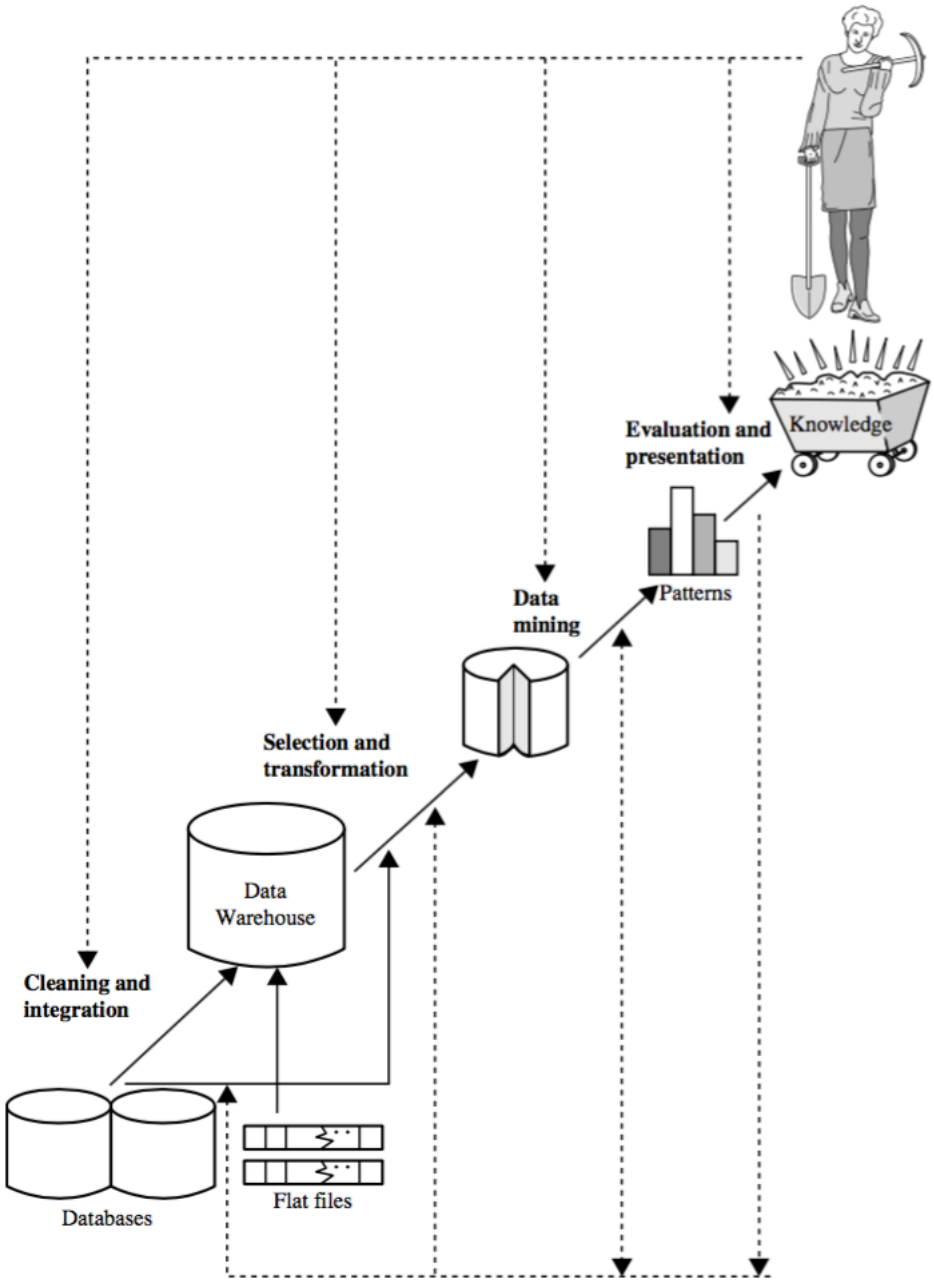

Figura 1: Mineração de dados como uma etapa no processo do KDD. Fonte: Han (2005)

2.2 Mineração de Dados

Como citado brevemente na introdução, *Data Mining* (mineração de dados) é o processo não trivial de extração de informações implícitas, anteriormente desconhecidas e potencialmente úteis a partir de uma fonte de dados (FRAWLEY; PIATETSKY-SHAPIRO; MATHEUS, 1992). A mesma, é uma parte integrante do processo de *Knowledge-Discovery in Databases (KDD)*, que utiliza técnicas estatísticas de descoberta de padrões através da análise do comportamento histórico dos dados.

Segundo Tan, Steinbach e Kumar (2005) a mineração de dados é o processo

automático de captura de informações úteis em grandes depósitos de dados. Esta possui alguns métodos que podem ser aplicados de acordo com uma finalidade específica. As tarefas a serem utilizadas no trabalho são classificação e regressão e estão descritas mais detalhadamente a seguir.

2.2.1 Classificação

A classificação é uma tarefa da mineração de dados que tem com objetivo buscar uma função que permita associar corretamente cada registro X_i de um base de dados a um único rótulo categórico Y_j, denominado classe, formando uma dupla (X,Y). A ferramenta *WEKA* possui uma coleção de algoritmos de aprendizado de máquina, na qual inclui os algoritmos da tarefa de classificação. Como existem muitos tipos de algoritmos de classificação, foram selecionados alguns para aplicar ao problema descrito neste trabalho.

2.2.2 Regressão

A tarefa de regressão compreende, fundamentalmente, na busca por funções, lineares ou não, que mapeiem os registros de uma base de dados em valores reais. Essa tarefa é similar à tarefa de classificação, sendo restrita apenas a atributos numéricos. Existem muitos tipos de algoritmos de regressão. Dessa forma, com base nos atributos da base de dados e do preprocessamento, foram selecionados alguns algoritmos para a tarefa em questão.

2.3 *WEKA*

A experiência mostra que nenhum esquema único de aprendizado de máquina é apropriado para todos os problemas de mineração de dados, e que o aprendizado universal é uma fantasia idealista segundo Witten, Frank e Hall (2011).

A ferramenta *Waikato Environment for Knowledge Analysis (WEKA)*[1], é uma coleção de algoritmos de aprendizado de máquina e de preprocessamento de dados. Além de possuir uma vasta coleção de algoritmos, desde algoritmos mais clássicos aos algoritmos mais atuais, possibilita a inclusão de novos algoritmos desde que atendam aos requisitos descritos na documentação.

De acordo com Witten, Frank e Hall (2011) o *WEKA* foi projetado para que seja possível a experimentação rápida de métodos existentes em novos conjuntos de dados de maneira flexível. O *WEKA* ainda fornece um suporte para todo o processo de mineração de dados experimental, como a preparação dos dados, avaliação sos resultados da aprendizagem de maneira estatística, além da visualização dos mesmos. Conta ainda com uma grande

[1] <http://www.cs.waikato.ac.nz/ml/weka/>

variedade de ferramentas de preprocessamento. Esse conjunto de ferramentas diversificadas e abrangentes é acessado por meio de uma interface comum, de modo que o usuário possa fazer uma comparação entre os diferentes métodos e identificar aqueles que são mais apropriados para o problema.

O arquivo de entrada de dados é o padrão *.arff* demonstrado na Figura 2, que apresenta um exemplo para o problema de dados meteorológicos. As linhas que começam com % são comentários. Seguindo os comentários no início do arquivo encontra-se o nome da relação, no caso *weather*. Em seguida, é apresentado um bloco {*@attribute* nome-do-atributo {possiveis-valores}}, onde a *tag* possíveis valores pode ser definida como *numeric, nominal*, ou ainda definida como classes discretas na primeira linha dos atributos da Figura 2. A partir da próxima linha tem-se uma declaração *@data* que é seguido das instâncias com os valores definidos nos atributos separados por vírgula.

```
% ARFF file for the weather data with some numeric features
%
@relation weather

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

@data
%
% 14 instances
%
sunny, 85, 85, false, no
sunny, 80, 90, true, no
overcast, 83, 86, false, yes
rainy, 70, 96, false, yes
rainy, 68, 80, false, yes
rainy, 65, 70, true, no
overcast, 64, 65, true, yes
sunny, 72, 95, false, no
sunny, 69, 70, false, yes
rainy, 75, 80, false, yes
sunny, 75, 70, true, yes
overcast, 72, 90, true, yes
overcast, 81, 75, false, yes
rainy, 71, 91, true, no
```

Figura 2: Exemplo de arquivo *.arff* para dados meteorológicos

2.4 Algoritmos

Os algoritmos de classificação e regressão utilizados neste trabalho são implementados e executados pela ferramenta *WEKA*, que é uma coleção de algoritmos de aprendizado de máquina e de preprocessamento de dados. Esta coleção inclui basicamente os algoritmos mais utilizados na literatura. Os algoritmos utilizados no trabalho podem ser divididos em: árvore de classificação, aprendizagem preguiçosa (*lazy*), baseado em funções e meta algoritmos.

A Tabela 1 apresenta uma breve explicação sobre os algoritmos de classificação utilizados.

Nome	Função
RandomCommitee	Cria um conjunto de classificadores básicos aleatorizados. A previsão final é uma média linear das previsões geradas pelos classificadores de base.
RandomTree	Constrói uma árvore que considera um dado número de atributos aleatórios em cada nó.
RandomForest	Cria várias árvores de classificação A floresta escolhe a classificação com maior pontuação.
KStar	Utiliza o método do Vizinho mais próximo com função de distância generalizada.
J48	Utiliza o algoritmo de árvore de decisão c4.5.
Bagging	Meta algoritmo que melhora a estabilidade e a precisão para reduz a variância. Geralmente aplicado a métodos de árvore de decisão.
REPTree	Método de árvore de decisão rápido que usa poda reduzida de erros.
RandomSubSpace	Constrói um conjunto de classificadores de base com diferentes conjuntos de atributos.
ClassificationRegression	Executa classificação usando um método de regressão.
LMT	Constrói modelos de árvore de regressão logística.
LogitBoost	Executa regressão logística aditiva.
SimpleLogistic	Constrói modelos de regressão logística linear com seleção de atributos embutida.
MultiClassClassifier	Usa um classificador de duas classes para conjuntos de dados multiclasse.
Ibk	Utiliza um classificador baseado em K-vizinhos mais próximos.
LWL	Algoritmo geral para aprendizagem localmente ponderada.
SMO	Algoritmo de otimização mínima sequencial para a classificação de vetores de suporte.

Tabela 1: Tabela de Algoritmos de Classificação utilizados. Modificada de (WITTEN; FRANK; HALL, 2011)

A Tabela 2 apresenta uma breve explicação sobre os algoritmos de regressão utilizados.

2.5 Métricas de Avaliação dos Algoritmos

As métricas de avaliação são utilizadas para identificar os melhores algoritmos a serem aplicados sobre o problema em questão. Para tal avaliação, os algoritmos da Tabela 1 e Tabela 2, utilizados para a descoberta de conhecimento em base de dados, são comparados de acordo com métricas baseadas em erros e correlação entre os dados. As métricas utilizadas são: coeficiente de correlação *(correlation coefficient - CC)*, média do erro absoluto *(Mean Absolute Error - MAE)*, raiz quadrada do quadrado do erro médio *(Root Mean Squared Error - RMSE)*, erro relativo absoluto *(Relative Absolute Error- RAE)*, raiz quadrada do erro relativo *(Root Relative Squared Error - RRSE)* e o classificações corretas *(Correctly Classified Instances - CCI)*. O melhor detalhamento das métricas será apresentado nas próximas subseções.

Nome	Função
Bagging	Meta algoritmo que melhora estabilidade e a precisão para reduzir a variância.
ResgressionByDiscretization	Executa a classificação usando métodos de regressão.
M5Rules	Obtém modelos de árvores executando o algoritmo M5.
AdditiveRegression	Método de regressão iterativo por ajustes dos ruídos.
SMOReg	Algoritmo de otimização mínima sequencial para a regressão de vetores de suporte.
MultilayerPerceptron	Redes neurais *backpropagation*.
LeastMedSq	Regressão robusta usando a mediana em vez da média.
RBFRegressor	Implementa uma rede de função de base radial.
IsotonicRegression	Constrói um modelo de regressão baseado em isotonia.
LWL	Algoritmo geral para aprendizagem localmente ponderada.
LinearRegression	Regressão linear múltipla baseada em padrão.
PaceRegression	Constrói modelos de regressão linear baseado no algoritmo *Pace*.
GaussianProcesses	Processo Gaussiano aplicado a regressão.
SimpleLinearRegression	Constrói um modelo de regressão linear com base em um único atributo.

Tabela 2: Tabela de Algoritmos de Regressão utilizados. Modificada de (WITTEN; FRANK; HALL, 2011)

2.5.1 Parâmetros de Avaliação dos Resultados do WEKA

Formalmente, o valor de θ é denotado como o valor verdadeiro de interesse, e $\hat{\theta}$ é denotado como valor estimado.

A métrica *Correlation coeficcient ou kappa statistic* (coeficiente de correlação) mostra a relação entre o valor verdadeiro e o valor estimado de uma classificação, com isso esse valor varia estritamente entre $-1 \geq \theta, \hat{\theta} \leq 1$. Quanto mais próximo de zero os valores de θ e $\hat{\theta}$ estão, a relação é basicamente inexistente e quanto mais próximo aos extremos a relação é muito forte, sendo, no caso negativo, uma relação muito forte inversa. A Figura 3 ilustra como essa métrica é uma das medidas mais comuns de erro de previsão.

A métrica MAE não leva em conta se o erro foi superestimado ou subestimado, caracterizando-se por ser a média dos erros cometidos pelo modelo de previsão durante uma série de execuções. Para calcular, subtrai-se o valor da previsão ao valor verdadeiro em cada período de execução. O resultado deverá sempre ser positivo, sempre em módulo, soma-se e divide-se pelo número de valores que é usado para obter a soma, representado formalmente pela equação (2.1):

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |\hat{\theta}_i - \theta_i| \tag{2.1}$$

A métrica *RMSE* também é usada como uma medida do erro de previsão. É determinada a soma dos erros de previsão ao quadrado e dividi-se pelo número de erros usado no cálculo. A *RMSE* pode ser expressa formalmente pela equação (2.2):

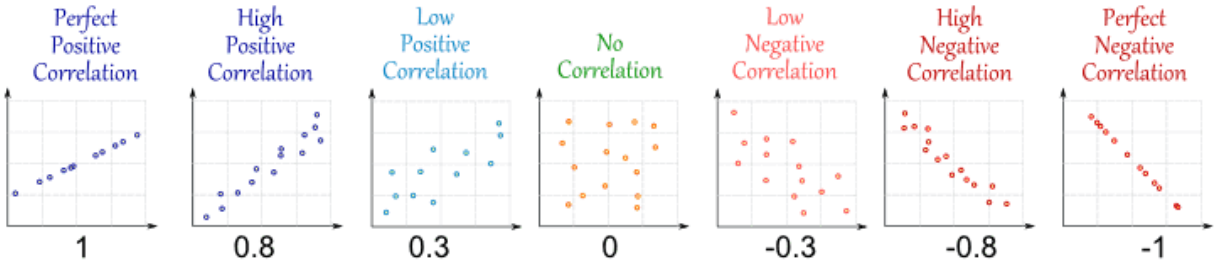

Figura 3: Gráfico da Métrica Coeficiente de Correlação

Fonte <http://www.mathsisfun.com/data/correlation.html>

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\hat{\theta}_i - \theta_i\right)^2} \tag{2.2}$$

Vale ressaltar que a diferença média entre MAE e $RMSE$ pode ser comparada em relação ao valor da variável.

A métrica $RRSE$, segundo Witten, Frank e Hall (2011) refere-se a uma medida um pouco diferente, onde o cálculo do erro é feito em relação ao que teria sido um preditor simples utilizado. O preditor simples em questão é apenas a média dos valores reais dos dados, denotados por 1. Assim, o $RRSE$ faz uma normalização, dividindo-se pelo erro quadrado total do indicador padrão, como pode ser visto na equação (2.3):

$$RRSE = \sqrt{\frac{\sum_{i=1}^{N}\left(\hat{\theta}_i - \theta_i\right)^2}{\sum_{i=1}^{N}\left(\overline{\theta} - \theta_i\right)^2}}, \text{sendo } \overline{\theta} \text{ a média dos valores de } \theta \tag{2.3}$$

A métrica RAE, segundo Witten, Frank e Hall (2011) é apenas o erro absoluto total, tendo o mesmo tipo de normalização do $RSSE$. Os erros são normalizados pelo erros do preditor simples que prevê os valores médios, de acordo com a equação (2.4):

$$RAE = \frac{\sum_{i=1}^{N}\left|\hat{\theta}_i - \theta_i\right|}{\sum_{i=1}^{N}\left|\overline{\theta} - \theta_i\right|}, \text{sendo } \overline{\theta} \text{ a média dos valores de } \theta \tag{2.4}$$

A métrica CCI, calculada a partir do resultado das classificações, mostra, em porcentagem, a quantidade de instâncias que foram corretamente classificadas.

Os atributos supracitados são utilizados para a tarefa de regressão e para a tarefa de classificação, onde são adicionados a porcentagem de instâncias que foram classificadas corretamente.

2.6 Programação Linear Inteira

A Programação Linear Inteira, também referida como Programação Inteira (PI), pode ser vista como um problema de Programação Matemática em que a função objetivo, bem como as restrições, são lineares, porém uma ou mais variáveis de decisão podem assumir apenas valores inteiros. O modelo formal de um problema de Programação Inteira pode ser expresso como:

$$\begin{aligned} Max\ (ou\ Min)\ z = \quad & c^T x + h^T y \\ Sujeito\ a: \quad & Ax + Gy \leq b \\ & x \geq 0,\ y \geq 0 \\ & x \in \Re^n,\ y \in \mathbb{Z}^p \end{aligned}$$

Neste modelo, x representa um conjunto de variáveis de decisão contínuas de dimensão n e y um conjunto de variáveis de decisão inteiras de dimensão p.

Os problemas de Programação Linear referem-se à distribuição eficiente de recursos limitados entre atividades competitivas, com a finalidade de atender a um determinado objetivo, por exemplo, maximização de lucros ou minimização de custos. Em se tratando de programação linear, esse objetivo será expresso por uma função linear, ao qual se dá o nome de função objetivo.

Para promover melhorias da função objetivo e tornar o problema viável é necessário atender uma série de restrições. Essas restrições são representadas por equações e inequações lineares. Uma vez obtido o modelo linear, constituído pela função objetiva (linear) e pelas restrições lineares, a programação linear se incumbe de achar a sua solução ótima (PUCCINI, 1980).

2.7 COIN-OR

Conforme visto anteriormente, uma forma de solução de problemas de PI se dá por meio da utilização do resolvedor de código aberto *COIN-OR Branch and Cut (CBC)*[2] (LOUGEE-HEIMER, 2003). Escrito na linguagem C++, o resolvedor possui um conjunto de bibliotecas que permitem a leitura, criação e manipulação de problemas dessa natureza. Assim, o *CBC* pode ser utilizado tanto como um resolvedor *stand-alone* quanto como uma biblioteca para auxiliar no desenvolvimento de algoritmos *branch-and-cut* customizáveis.

O *CBC* possui uma série diversificada de parâmetros, que podem influenciar diretamente no desempenho da resolução de um problema. Dessa forma, a tarefa de

[2] <https://projects.coin-or.org/Cbc>

identificar e utilizar os melhores parâmetros para um dado problema é complexa, porém essencial. Tal situação pode ser resolvida por meio da aplicação do processo de Descoberta de Conhecimento em Base de Dados detalhado na subseção 2.1.

Neste trabalho são utilizadas técnicas de descoberta de padrões através da análise do comportamento histórico dos dados, possibilitando que novas instâncias de problemas sejam executadas de forma eficiente no CBC.

2.8 Trabalhos Relacionados

O trabalho de Šilc, Taškova e Korošec (2015), desenvolve um modelo utilizando a tarefa de regressão multi-nível para tentar melhorar o desempenho de um algoritmo meta-heurístico, ajustando parâmetros para o *Differential Ant-Stigmergy Algotithm (DASA)*. Este algoritmo foi inspirado no comportamento de auto-organização eficiente das colônias de formigas através da comunicação por ferormônio, conhecida como *Stigmergy*. O principal desafio do projeto está em fornecer uma configuração para um determinado tipo de problema para o algoritmo *DASA*. Apesar da escolha *default* ser uma boa escolha de configuração inicial, esta pode resultar em soluções de baixa qualidade em um problema de otimização específico. O ajuste de parâmetros de um algoritmo pode levar a um melhor desempenho, no entanto, é computacionalmente caro. Na prática, os parâmetros do algoritmos são aperfeiçoados a fim de obter um melhor desempenho do algoritmo para o problema da esfera com 5000 definições em relação a duas dimensões, de 20 e 40. O problema foi formulado como uma tarefa de modelagem preditiva usando árvores de decisão para modelar os valores de erros da função em termo dos valores dos parâmetros do *DASA*. Como as variáveis de erro da função são contínuos, a tarefa utilizada é a regressão.

O trabalho de Lessmann, Caserta e Arango (2011), contribui para a literatura propondo uma solução baseada em mineração de dados para o problema de ajuste de parâmetros de meta-heurísticas e providenciando evidências empíricas da sua eficiência. Para este fim, o estudo de caso sistematicamente emprega a exploração da viabilidade de previsão efetiva de configurações de parâmetros para uma meta-heurística, o algoritmo *Particle Swarm Optimization (PSO)*. O estudo é estabelecido com métodos de regressão para entender a natureza da relação entre as variáveis independentes empregadas e valores de parâmetros efetivos (ex., linear versus não linear) e identificar modelos candidatos promissores. Desde incorporação de modelos de previsão como agentes de ajuste no *PSO* – ou qualquer outro meta-heurística – iria envolver lotes de dados que iriam se tornar disponíveis ao longo de sucessivas iterações do *PSO*. Uma análise da curva de aprendizagem é conduzido para explorar a sensibilidade do tamanho dos dados e simular o aprendizado online do modelo de previsão. Isso complementa a avaliação de diferentes modelos candidatos e cria um embasamento para pesquisas futuras para desenvolver

e avaliar uma meta-heurística híbrida com um agente de ajuste integrado. O trabalho propõe uma nova perspectiva no trabalho de ajuste das configurações de parâmetros das meta-heurísticas. A abordagem é genérica em relação aos seus princípios e tem sido avaliado empiricamente em conjunção com o algoritmo *PSO* em particular. Os resultados empíricos indicam que a eficácia das configurações dos parâmetros pode realmente ser relacionada a um problema especí fico de informação (ex. assinatura de partículas), para que o método de ajuste baseado em regressão aparenta ser promissor em geral.

3 Análise e Desenvolvimento

Nesta seção serão explicados o preprocessamento dos dados, a avaliação dos algoritmos, a seleção dos mesmos, além da apresentação da biblioteca criada.

3.1 Preprocessamento de Dados

O preprocessamento é parte de preparação dos dados para a mineração de dados. Devido a utilização da ferramenta *WEKA* para fazer os testes dos diversos algoritmos de regressão e classificação, a base de dados teve de ser transformada para o formato *.arff*, um padrão definido pela *WEKA* para inserir os dados na ferramenta.

Inicialmente foi estudada a base de dados utilizada pelo *COIN-OR Branch and Cut (CBC)* do Grupo de Otimização de Algoritmos (GOAL/UFOP) da Universidade Federal de Ouro Preto que desenvolve pesquisas nas áreas de otimização mono e multi-objetivo, inteligência computacional, mineração de dados, tratamento e recuperação de informação e algoritmos. Como foi dito na introdução, as aplicações dos projetos do GOAL/UFOP envolve uma variedade de problemas do mundo real.

A base de dados está organizada em um arquivo de formato ".csv" onde cada atributo /característica é separado por vírgula, onde cada linha corresponde a uma instância de um problema genérico executado no *COIN-OR* com todas as suas características, contendo um total de 3184 linhas. O conjunto de registros é representado pelas instâncias que possuem os atributos dos problemas de otimização combinatória e linear executados no *CBC*. Os atributos utilizados são relacionados aos principais componentes de um problema de PI:

- **variáveis**: contém atributos que apresentam o número total de variáveis presentes no problema, bem como o detalhamento da quantidade de variáveis binárias, inteiras e contínuas;
- **restrições**: contém atributos que contabilizam o número total de restrições do problema e os principais tipos de restrições (*set partition*, *set packing*, *set cover*, *cardinality*, *invariant knapsack*, *knapsack*, *binary flow conservation*, *integer flow conservation* e *continuous flow conservation*);
- **matriz de incidência**: contém os valores mínimo e máximo dos coeficientes da matriz de incidência do problema;
- **conjunto de parâmetros**: contém uma *string* que é o conjunto de parâmetros a ser utilizado, passado como parâmetro para execução do problema no resolvedor CBC.

Além dos atributos das instâncias, temos a lista de alguns parâmetros que podem ser utilizados pelo resolvedor *CBC*. Abaixo segue a lista de parâmetros utilizados neste trabalho:

- **diveopt**: ativa ou desativa as heurísticas de mergulho (*diving*). No caso da ativação, decide em quais variáveis essas heurísticas serão aplicadas;
- **zero**: ativa ou desativa os cortes *Zero-Half* ($0/\frac{1}{2}$) (CAPRARA; FISCHETTI, 1996). No caso da ativação, decide em quais ramos da árvore esses cortes serão gerados e inseridos;
- **strong**: número máximo de variáveis candidatas para aplicar *strong branching* (ACHTERBERG; KOCH; MARTIN, 2005), por padrão, o valor desse parâmetro é igual a 5;
- **trust**: número máximo de ramificações utilizando seleção de variáveis por *strong branching*, por padrão, o valor desse parâmetro é igual a 5;
- **tunep**: parâmetro usado para controlar a etapa de pré-processamento, que inclui estratégias para alterar coeficientes e fixar os limites de algumas variáveis, diminuir o valor do lado direito de algumas restrições, etc;
- **proximity**: ativa ou desativa a heurística *Proximity Search* (FISCHETTI; MONACI, 2014).

A classe considerada neste trabalho representa o *GAP*, medida percentual da distância entre a solução encontrada e a solução ótima. Sendo o valor do *GAP* quando mais próximo de 1 melhor é a seleção de parâmetros. Assim, será identificada a melhor relação entre os atributos das instâncias e as configurações dos parâmetros que se aproxime do *GAP* ideal, próximo a zero.

3.1.1 Limpeza e Padronização dos Dados

De maneira similar foram definidos dois arquivos *.arff* sendo diferentes apenas em dois atributos: o *rank* que é o atributo de classe e os parâmetros que, para as tarefas de classificação, são definidos como nominal, e, para as tarefas de regressão, são definidos como numérico.

Como mostra as Figuras 4 e 5, as modificações foram feitas através do *WEKA* utilizando os filtros. A diferença, apesar de pequena, é de suma importância para que os algoritmos de regressão tenham efetividade. Nessa tarefa, cria-se uma função matemática que, dados os valores numéricos de novas instâncias, retorna o valor numérico da classe

(aqui chamada de *rank*). Já na classificação, os campos de parâmetros e o *rank* são nominais, ou seja, não possuem um valor numérico atribuído, mesmo que indicado como numérico.

O mapeamento citado anteriormente de nominal para numérico foi feito apenas de forma simples efetuando uma categorização direta de acordo com a posição de entrada, ou seja o primeiro item do parâmetro foi colocado como o número 1, o segundo item como número 2 e assim sucessivamente, e para o atributo *rank* que é o classificador como ele já é numérico por padrão apenas foi transformado em nominal de forma direta, sendo o numeral 1 transformado em categoria 1.

```
@relation cbc-data

% attributes variables
@attribute nome         string
@attribute totalVar     numeric
@attribute binary       numeric
@attribute int          numeric
@attribute continuous   numeric
% attributes constraints
@attribute totalCons    numeric
@attribute part         numeric
@attribute pack         numeric
@attribute cov          numeric
@attribute card         numeric
@attribute iknp         numeric
@attribute knp          numeric
@attribute bflow        numeric
@attribute iflow        numeric
@attribute mflow        numeric
% attributes matrix A
@attribute min          numeric
@attribute max          numeric
% attributes experiment
@attribute parameter    {cbc2210ostbzerodiveopt7sb15,cbc2210stbsb10zero,
cbc2210ostbsb10-20zero,cbc2210ostbsb20-40zero,cbc2210ostbsb20-30zero,
cbc2210ostbzerodiveopt3,cbc2210ostbzerodiveopt3sb,cbc2210stbsb15zero,
cbc2210ostbzerodiveopt5,cbc2210ostbsb20-20zero,cbc2210ostbzerodiveopt6,
cbc2210stb,cbc2210ostbzerodiveopt4,cbc2210ostbzerodiveopt7,
cbc2210ostbsb10-10zero,cbc2210ostbzerodiveopt7sb}
%@attribute rank        numeric
@attribute class        {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17}

@data
flugpl,18,0,11,7,18,0,0,0,0,0,0,0,4,1,-100,150,cbc2210ostbzerodiveopt7sb15,2
flugpl,18,0,11,7,18,0,0,0,0,0,0,0,4,1,-100,150,cbc2210stbsb10zero,1
```

Figura 4: Exemplo de arquivo *.arff* para dados na classificação

A partir do processo de criação do arquivo *.arff* o mesmo foi adicionado ao *WEKA* para o preprocessamento, retirando o nome. Após a limpeza nos dados e da diferenciação na definição dos arquivos *.arff*, passa-se para etapa de aplicação das tarefas de mineração de dados, sendo de classificação e regressão. A base de dados foi utilizada tanto para o teste quanto para o treinamento. Os algoritmos, bem como as métricas que foram utilizadas para avaliação dos algoritmos estão detalhadas nas subseções 2.4 e 2.5.

```
@relation cbc-data

% attributes variables
@attribute nome         string
@attribute totalVar     numeric
@attribute binary       numeric
@attribute int          numeric
@attribute continuous   numeric
% attributes constraints
@attribute totalCons    numeric
@attribute part         numeric
@attribute pack         numeric
@attribute cov          numeric
@attribute card         numeric
@attribute iknp         numeric
@attribute knp          numeric
@attribute bflow        numeric
@attribute iflow        numeric
@attribute mflow        numeric
% attributes matrix A
@attribute min          numeric
@attribute max          numeric
% attributes experiment
@attribute parameter    numeric
%@attribute rank        numeric
@attribute class        numeric

@data
rgn,180,100,0,80,24,0,4,0,0,0,0,0,0,20,-4.6,1,1,1
br3,6368,6368,0,0,12056,13,520,4,56,345,0,0,0,0,-1,1,1,6
nw04,87482,87482,0,0,36,36,0,0,0,0,0,0,0,0,1,1,1,3
egout,141,55,0,86,98,0,0,0,0,0,0,0,0,43,-117.04,1,1,1
```

Figura 5: Exemplo de arquivo *.arff* para dados na regressão

3.2 Avaliação de Desempenho dos Algoritmos

A escolha dos algoritmos utilizados como estudo de caso neste trabalho foi realizada de modo a exercitar as principais técnicas relacionadas com a tarefa de classificação e regressão. Vale ressaltar que foram avaliados outros algoritmos disponíveis no *WEKA*, mas não são reportados por não terem obtido resultados relevantes. Essa seleção foi feita com base na análise das tabelas 3 e 4, que mostram os desempenhos dos algoritmos de regressão e classificação, respectivamente.

A partir das tabelas é possível concluir que o problema obtém melhores resultados com a tarefa de classificação. Considerando apenas os melhores algoritmos de classificação que obtiveram melhores resultados, ou seja, menores erros, tem-se o *RandomCommitte*, *RandomTree* e o *RandomForest*. Eles possuem erros bem menores que os demais algoritmos, apresentam um coeficiente de correlação (CC) próximo a 1, (ver Subseção 2.5) e classificam corretamente 98% das instâncias executadas. Tais algoritmos trabalham com fatores aleatórios e possuem melhor desempenho que um algoritmo que trabalha de forma de sequencial, como por exemplo, o *SMO*.

Algoritmo	CC	MAE	RMSE	RAE (%)	RRSE (%)
Bagging	0.80	1.138	1.681	51.516	61.026
RegressionByDiscretization	0.77	1.224	1.773	55.400	64.356
M5Rules	0.65	1.535	2.102	69.442	76.312
AdditiveRegression	0.40	1.995	2.532	90.280	91.922
SMOReg	0.16	2.013	2.817	91.076	102.257
MultilayerPerceptron	0.32	2.030	2.779	91.844	100.891
LeastMedSq	0.11	2.092	3.010	94.662	109.256
RBFRegressor	0.24	2.118	2.675	95.865	97.100
IsotonicRegression	0.20	2.136	2.702	96.679	98.076
LWL	0.20	2.140	2.702	96.846	98.074
LinearRegression	0.20	2.145	2.697	97.062	97.897
PaceRegression	0.20	2.150	2.697	97.275	97.900
GaussianProcesses	0.15	2.175	2.725	98.427	98.911
SimpleLinearRegression	0.07	2.199	2.748	99.509	99.765

Tabela 3: Tabela de Comparação dos Algoritmos de Regressão

Algoritmo	CC	MAE	RMSE	RAE(%)	RRSE(%)	CCI(%)
RandomCommitee	0.980	0.002	0.032	2.076	14.325	98.30
RandomTree	0.979	0.002	0.033	2.194	14.817	98.18
RandomForest	0.980	0.028	0.090	28.090	40.233	98.30
LazyKStar	0.885	0.045	0.129	44.800	57.699	90.23
J48	0.578	0.056	0.167	55.814	74.730	64.45
Bagging	0.556	0.065	0.172	65.177	76.959	62.69
REPTree	0.461	0.069	0.186	69.390	83.325	54.55
RandomSubSpace	0.473	0.072	0.183	71.764	81.964	55.75
AttributeSelected	0.428	0.072	0.190	72.446	85.140	51.79
ClassificationRegression	0.415	0.080	0.193	79.577	86.385	51.22
LMT	0.313	0.080	0.205	80.279	91.605	43.59
IterativeOptimizer	0.239	0.091	0.210	90.882	94.019	39.35
LogitBoost	0.239	0.091	0.210	90.882	94.019	39.35
FilteredClassifier	0.116	0.093	0.216	92.948	96.437	32.60
SimpleLogistic	0.071	0.096	0.218	95.688	97.670	29.40
MultiClassClassifier	0.064	0.096	0.218	95.815	97.757	28.89
Ibk	0.066	0.097	0.220	97.346	98.558	28.83
MultiClassUpdateable	0.019	0.098	0.222	97.735	99.240	27.07
LWL	0.005	0.099	0.222	99.055	99.459	26.22
SMO	0.046	0.106	0.228	106.114	102.011	28.67

Tabela 4: Tabela de Comparação dos Algoritmos de Classificação

Na Figura 6, é apresentado o modelo simplificado da árvore de classificação gerada pelo algoritmo *Random Tree*, apresentando somente os nós iniciais, visto que o modelo é extenso. O algoritmo que obteve o melhor desempenho, *Random Committee*, utiliza como classificador o *Random Tree* e encontra a média final das previsões geradas por esse

classificador variando somente as bases.

Podemos observar que os nós principais correspondem aos atributos e quantidade de variáveis que o problema possui, divididos em dois grupos: os que possuem valores menores que 552 e maiores ou iguais a 552. A partir do nó raiz da árvore expande-se uma sequência de sub-árvores, diferenciando os demais atributos em duas ou mais ramificações até que seja concluído a classificação. Os dois filhos diretos da raiz são *mflow* e *min*. O primeiro, *mflow*, é um atributo relacionado a um tipo de restrição que pode envolver variáveis inteiras e contínuas, cujo objetivo é garantir a conservação do fluxo em uma rede. O segundo, *min*, é o menor coeficiente da matriz do problema. Com isso, consegue-se perceber uma grande variação nas características das instâncias para o problema da recomendação de parâmetros, tornando impossível a percepção de algum conhecimento por meio desses atributos diretamente.

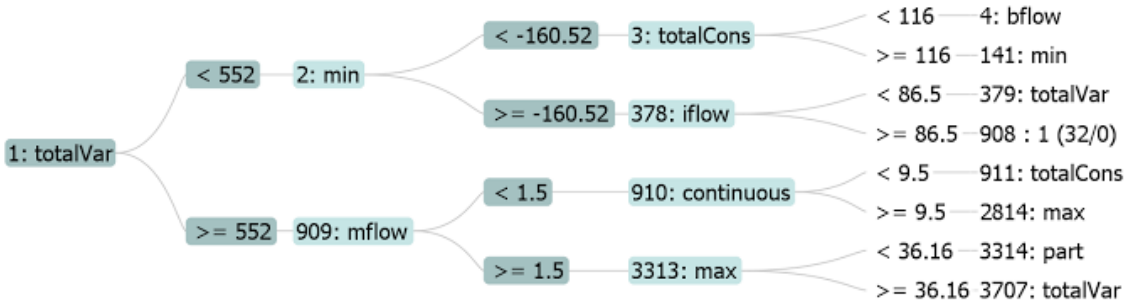

Figura 6: Modelo Simplificado da Árvore gerada pelo *Random Tree*

3.3 Biblioteca

A partir da execução e seleção dos melhores algoritmos, foi desenvolvido uma aplicação em Java, utilizando a biblioteca do *WEKA* para que seja possível a classificação de novas instâncias. Essa biblioteca contém basicamente uma interface via terminal, devido ao fato dos utilizadores da biblioteca já estarem familiarizados com a tela do *prompt* e não com projetos de interfaces gráficas de usuários.

A aplicação consiste em um módulo de treinamento e um módulo de classificação. O módulo de treinamento tem como objetivo principal prover melhorias para a classificação para o problema descrito neste trabalho. O módulo de classificação tem como principal objetivo classificar novas instâncias do problema a partir da utilização de modelos já treinados.

3.3.1 Pseudocódigo

O Algoritmo 1 mostra basicamente a influência e a dependência da biblioteca *WEKA* para a classificação e regressão, além de, ainda, ser utilizada na limpeza dos dados

(ver 3.1.1). O pseudocódigo consiste em um processo que faz a seleção através das entradas fornecidas pelos argumentos que são tratadas pelo *ArgParser4j* [1], que trabalha com os argumentos passados como parâmetros no terminal na execução da biblioteca.

O pseudocódigo é dividido em duas funções, de classificação e treinamento, que neste trabalho serão tratadas como módulos separadamente. O módulo de classificação recebe como parâmetros o modelo, a base de teste e o nome do arquivo de saída. Já o módulo de treinamento recebe, além dos parâmetros como o modelo, a base de dados de teste, nome do arquivo de saída e a base de treinamento, sendo estes argumentos para ambos módulos recebidos parcialmente como argumentos da biblioteca.

A partir da seleção dos argumentos recebidos pelo *ArgParser4j*, faz-se o teste para selecionar qual módulo executar. No módulo de classificação, assim como no módulo de treinamento, são executados dois passos iniciais. Primeiramente, é realizada a aplicação do filtro para remoção do nome e a colocação do atributo *rank* como classe para o problema de classificação. Em seguida, escreve-se em um arquivo de saída as instâncias classificadas, substituindo a classe do arquivo de entrada pela classe prevista pelo modelo. A partir do terceiro passo, o módulo de classificação efetua a leitura do modelo salvo na pasta da biblioteca. O algoritmo de classificação pode ser escolhido entre os três melhores. Com isso, classifica-se cada instância separadamente, aplicando o modelo de classificação escolhido previamente. Já o módulo de treinamento usa o *evaluation*, que avalia a base de teste. Feito isso, será definido o classificador para a base de testes, afim de validar o modelo treinado e escrever a solução em um arquivo .model.

A biblioteca possui um argumento especifico, que será exibido quando algum argumento não estiver de acordo com o estabelecido, exibindo essa tela com o intuito de ajudar no uso da biblioteca. A Figura 7 é um exemplo da tela de ajuda exibida na tela quando passado por argumento “-h“ ou através de algum erro nos argumentos.

3.3.2 Módulo de Treinamento

O módulo de treinamento foi elaborado com intuito de fazer com que a aplicação possa ser utilizada a partir de novos modelos treinados. Com isso, caso a base seja tão bem elaborada quanto a base de dados utilizada neste trabalho, pode-se haver melhorias no modelo para possíveis classificações posteriores. O módulo de treinamento vai ser executado a partir de um arquivo *.arff*, contendo as instâncias de treinamento. O usuário selecionará um algoritmo dentre os três melhores classificados para a tarefa de classificação, sendo eles o *RandomTree, RandomCommittee* e o *Random Forest*. Por fim, é salvo o modelo treinado em um arquivo com a extensão *.model* e de nome indicando qual algoritmo foi utilizado, sendo esse o padrão para que o módulo de classificação consiga utilizar estes modelos para classificar novas instâncias.

[1] https://argparse4j.github.io/

```
λ java -jar rp-cbc.jar -h
usage: rp-cbc [-h] [--train algorithm[rtree, rforest, rcommit] infile]
              [--class [rtree,rforest,rccommit].model test_file classsify_file]

Classifier CBC parameters.

optional arguments:
  -h, --help            show this help message and exit
  --train algorithm[rtree, rforest, rcommit] infile
                        [algorithm] - Default select algorithm
                        [infile] - File instances training
  --class [rtree,rforest,rccommit].model test_file classsify_file
                        Default model classify
                        File instances test
                        File classified instances
```

Figura 7: Tela de ajuda da biblioteca

É possível visualizar o fluxo de execução do módulo de treinamento através da Figura 8. Para que o fluxo seja completo e válido é preciso que as instâncias de treinamento sejam disponibilizadas para que o modelo seja gerado.

Figura 8: Fluxo de execução do módulo de treinamento

Para executar a biblioteca é necessário a instalação do *Java Jdk* [2] e executar a partir do terminal de comandos do sistema operacional o seguinte comando: "java -jar rp-cbc.jar –train <nome-do-algoritmo(rtree, rforest, rcommit)> <nome-da-base-de-dados.arff>", apresentado de acordo com a Figura 9 .

```
λ java -jar rp_cbc.jar --train rtree cbc-exp-mine.arff
Training
Successfully created model!
```

Figura 9: Tela de execução do módulo de treinamento

[2] http://www.oracle.com/technetwork/pt/java/javase

Algorithm 1 rp-cbc algorithm

Require: Java
Require: Weka
Require: ArgsParse4j

```
1: procedure RPCBC(args[])
2:     if train ∈ args then
3:         CLASSIFY(models, training, filename)
4:     else if class ∈ args then
5:         TRAINING(models, training, testing, filename)
6:     else
7:         HELP(message)                          ▷ Mensagem de ajuda de uso -h
8:     end if
9: end procedure

10: function CLASSIFY(model, testing, filename)
11:     removeType(testing)                                  ▷ Remove o nome
12:     setClass(testing)                          ▷ Coloca o rank como classe
13:     readModel(model)
14:     for i ∈ testing do
15:         classified ← classifyInstance(i)
16:     end for
17:     write(filename, classified)
18: end function

19: function TRAINING(models, training, testing, filename)
20:     RemoveType(training, testing)                        ▷ Remove o nome
21:     SetClass(training, testing)                ▷ Coloca o rank como classe
22:     evaluation(training)
23:     buildClassifier(training)
24:     evaluateModel(models, testing)
25:     write(filename, models)
26: end function
```

3.3.3 Módulo de Classificação

O módulo de classificação foi elaborado com intuito de classificar novas instâncias a partir dos modelos de classificação treinados, que são baseados nos três melhores algoritmos de classificação para esse problema: *RandomTree, RandomForest e RandomCommittee*. O módulo de classificação vai executar a partir da escolha de um dos modelos treinados e de um arquivo *.arff* com as instâncias sem classe, utilizando no lugar do atributo classe o caractere "?". Além disso, há a necessidade de incluir um nome para o arquivo que será criado com as classificações das instâncias, que de forma simplificada vai aplicar o algoritmo e trocar o caractere interrogação "?"pela respectiva classe retornada pelo algoritmo.

Já o fluxo de execução do módulo de classificação pode ser visualizado através da Figura 10. A principal diferença entre o fluxo de classificação e o fluxo de treinamento é

que a entrada agora é representada por instâncias a serem classificadas, ou seja, que não possuem classes e a saída é justamente a classe para essas novas instâncias. Os algoritmos selecionados para classificação utilizarão o modelo uma vez treinado.

Figura 10: Fluxo de execução do módulo de Classificação

Para executar é necessário a instalação do *Java Jdk* e executar a partir do terminal de comandos do sistema operacional o seguinte comando: "java -jar rp-cbc.jar –class <nome-do-algoritmo(rtree, rforest, rcommit)> <instancias-sem-classe.arff> <saida-com-classficação.arff>", apresentado de acordo com a Figura 11 .

```
λ java -jar rp_cbc.jar --class rtree cbc-test.arff saida.txt
Instances successfully classified!
```

Figura 11: Tela de execução do módulo de Classificação

4 Conclusão e Trabalhos Futuros

Neste trabalho foi apresentado um estudo baseado na mineração de dados, mais especificamente com as tarefas de classificação e regressão. A tarefa de classificação mostrou-se mais eficiente para a recomendação de parâmetros a serem utilizados pelo *COIN-OR Branch and Cut* na resolução de problemas de Programação Inteira. Nos experimentos realizados sobre uma base de dados diversificada foi possível obter uma taxa de acerto de 98% por meio da utilização de algoritmos baseados em árvores de decisão (*Random Commitee*, *RandomForest* e *Random Tree*). Com uma boa acurácia, é possível inferir qual conjunto de parâmetros associado ao *CBC* levaria à melhor solução possível para um dado problema.

Vale ressaltar que existe uma limitação quanto à recomendação dos parâmetros: foram selecionadas e utilizadas dezesseis combinações entre um vasto conjunto de parâmetros. Nesse sentido, destacam-se como trabalhos futuros a inclusão de novas combinações de parâmetros, a fim de ampliar a base de dados, refinar os resultados obtidos e obter uma boa taxa de acerto para novas instâncias. Além disso, pretende-se desenvolver uma aplicação que faça a recomendação dos parâmetros de forma independente da ferramenta *WEKA*, para que esse processo possa ser integrado diretamente com o uso do *CBC*. Por fim, pretende-se investigar a importância de cada atributo no processo de classificação, especialmente cada parâmetro e sua influência direta na produção de uma solução de qualidade.

Referências

ACHTERBERG, T.; KOCH, T.; MARTIN, A. Branching rules revisited. *Operations Research Letters*, v. 33, n. 1, p. 42 – 54, 2005. ISSN 0167-6377. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167637704000501>. Citado na página 35.

CAPRARA, A.; FISCHETTI, M. {0, 1/2}-chvátal-gomory cuts. *Mathematical Programming*, v. 74, n. 3, p. 221–235, 1996. ISSN 1436-4646. Disponível em: <http://dx.doi.org/10.1007/BF02592196>. Citado na página 35.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. The kdd process for extracting useful knowledge from volumes of data. *Commun. ACM*, ACM, New York, NY, USA, v. 39, n. 11, p. 27–34, nov. 1996. ISSN 0001-0782. Disponível em: <http://doi.acm.org.ez28.periodicos.capes.gov.br/10.1145/240455.240464>. Citado 2 vezes nas páginas 21 e 24.

FISCHETTI, M.; MONACI, M. Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics*, v. 20, n. 6, p. 709–731, 2014. ISSN 1572-9397. Disponível em: <http://dx.doi.org/10.1007/s10732-014-9266-x>. Citado na página 35.

FRAWLEY, W. J.; PIATETSKY-SHAPIRO, G.; MATHEUS, C. J. Knowledge discovery in databases: An overview. *AI Mag.*, American Association for Artificial Intelligence, Menlo Park, CA, USA, v. 13, n. 3, p. 57–70, set. 1992. ISSN 0738-4602. Disponível em: <http://dl.acm.org/citation.cfm?id=140629.140633>. Citado na página 25.

HAN, J. *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005. ISBN 1558609016. Citado 4 vezes nas páginas 13, 21, 24 e 25.

LESSMANN, S.; CASERTA, M.; ARANGO, I. M. Tuning metaheuristics: A data mining based approach for particle swarm optimization. *Expert Systems with Applications*, v. 38, n. 10, p. 12826 – 12838, 2011. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417411005914>. Citado na página 32.

LOUGEE-HEIMER, R. The common optimization interface for operations research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, v. 47, n. 1, p. 57–66, Jan 2003. ISSN 0018-8646. Citado 2 vezes nas páginas 21 e 31.

PUCCINI, A. d. L. *Introdução a Programação linear*. Rio de Janeiro, RJ, BR: Livros técnicos e científicos editora S.A, 1980. Citado na página 31.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005. ISBN 0321321367. Citado na página 25.

WITTEN, I. H.; FRANK, E.; HALL, M. A. *Data Mining: Practical Machine Learning Tools and Techniques*. 3rd. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers

Inc., 2011. ISBN 0123748569, 9780123748560. Citado 5 vezes nas páginas 15, 26, 28, 29 e 30.

ŠILC, J.; TAšKOVA, K.; KOROšEC, P. Data mining-assisted parameter tuning of a search algorithm. *Informatica (03505596)*, v. 39, n. 2, p. 169 – 176, 2015. ISSN 03505596. Disponível em: <http://search-ebscohost-com.ez28.periodicos.capes.gov.br/login.aspx?direct=true&db=iih&AN=108472080&lang=pt-br&site=ehost-live&authtype=ip,cookie,uid>. Citado na página 32.

APÊNDICE A – Artigo aceito no SBPO XLVIII

Recomendação de parâmetros para o *COIN-OR Branch and Cut*

Rafael de Sousa Oliveira Martins
Departamento de Computação e Sistemas
Rua Trinta e seis, 115 – Bairro Loanda – CEP 35931-008 – João Monlevade – MG – Brasil
martins.rso@gmail.com

Janniele Aparecida Soares Araujo
Departamento de Computação e Sistemas
Rua Trinta e seis, 115 – Bairro Loanda – CEP 35931-008 – João Monlevade – MG – Brasil
janniele@decsi.ufop.br

Samuel Souza Brito
Departamento de Computação e Sistemas
Rua Trinta e seis, 115 – Bairro Loanda – CEP 35931-008 – João Monlevade – MG – Brasil
samuelbrito@decsi.ufop.br

Haroldo Gambini Santos
Departamento de Computação
Rua Quatro, 786 – Bairro Bauxita – CEP 35400-000 – Ouro Preto – MG - Brasil
haroldo@iceb.ufop.br

RESUMO

Este trabalho propõe a recomendação de parâmetros para o resolvedor *COIN-OR Branch and Cut (CBC)*, com o intuito de obter a melhor configuração de parâmetros para problemas de otimização e melhorar a eficiência na busca da solução ótima. Tal situação pode ser resolvida por meio da aplicação do processo de Descoberta de Conhecimento em Base de Dados (*Knowledge Discovery in Databases - KDD*). Os resultados obtidos foram satisfatórios, conseguiu-se encontrar um modelo genérico em árvore, para diferentes tipos de problemas, em que a indicação de parâmetros seja feita de forma confiável.

PALAVRAS CHAVE. COIN-OR Branch and Cut. Otimização Combinatória. Recomendação de Parâmetros. Descoberta de conhecimento em Base de Dados.

Tópicos (indique, em ordem de PRIORIDADE, o(s) tópicos(s) de seu artigo)

ABSTRACT

This paper proposes the recommendation of parameters to the resolver COIN-OR Branch and Cut (CBC), in order to get the best parameter settings for optimization problems and improve efficiency to find the optimal solution. This situation can be resolved by the application of process Knowledge Discovery in Databases (KDD). The results were satisfactory, it was possible to find a generic model tree for different types of problems, in which the parameter indication is made in way trustworthy.

KEYWORDS. COIN-OR Branch and Cut. Combinatorial Optmization. Recommendation parameters. Knowledge Discovery in Databases.

Paper topics (indicate in order of PRIORITY the paper topic(s))

1. Introdução

A Programação Linear Inteira, também referida como Programação Inteira (PI), pode ser vista como um problema de Programação Matemática em que a função objetivo, bem como as restrições, são lineares, porém uma ou mais variáveis de decisão podem apenas assumir valores inteiros. O modelo formal de um problema de Programação Inteira pode ser expresso como:

$$\begin{aligned} Max\ (ou\ Min)\ z = \quad & c^T x + h^T y \\ Sujeito\ a: \quad & Ax + Gy \leq b \\ & x \geq 0,\ y \geq 0 \\ & x \in \Re^n,\ y \in \mathbb{Z}^p \end{aligned}$$

Neste modelo, x representa um conjunto de variáveis de decisão contínuas de dimensão n e y um conjunto de variáveis de decisão inteiras de dimensão p.

Uma forma de solução de problemas de PI se dá por meio da utilização do resolvedor de código aberto *COIN-OR Branch and Cut* (CBC)[1] [Lougee-Heimer, 2003]. Escrito na linguagem C++, o resolvedor possui um conjunto de bibliotecas que permitem a leitura, criação e manipulação de problemas dessa natureza. Assim, o CBC pode ser utilizado tanto como um resolvedor *stand-alone* quanto como uma biblioteca para auxiliar no desenvolvimento de algoritmos *branch-and-cut* customizáveis.

O CBC possui uma série diversificada de parâmetros, que podem influenciar diretamente no desempenho da resolução de um problema. Dessa forma, a tarefa de identificar e utilizar os melhores parâmetros para um dado problema é complexa, porém essencial. Tal situação pode ser resolvida por meio da aplicação do processo de Descoberta de Conhecimento em Base de Dados (*Knowledge-Discovery in Databases - KDD*), que segundo Fayyad et al. [1996] é o processo que visa o descobrimento de conhecimento válido, relevante e desconhecido em base de dados. Esse processo é dividido em 5 etapas: seleção, pré-processamento, transformação, mineração de dados e interpretação.

Neste trabalho são utilizadas técnicas de descoberta de padrões através da análise do comportamento histórico dos dados, possibilitando que novas instâncias de problemas sejam executadas de forma eficiente no CBC.

2. O problema

Para que o resolvedor CBC possa obter resultados satisfatórios na solução de problemas de PI é preciso que sejam definidos bons parâmetros para cada problema de entrada. A escolha de parâmetros pode influenciar diretamente no desempenho do resolvedor, o que torna esse passo uma atividade essencial na construção da solução.

O objetivo deste trabalho é a recomendação de um conjunto parâmetros a ser utilizado pelo CBC na resolução de problemas de PI. Para isso, se faz necessário o conhecimento prévio de uma base de dados diversificada, analisar as características de problemas já executados pelo CBC e construir um modelo de classificação confiável.

Em problemas de otimização podemos considerar o *GAP* como uma medida da qualidade de uma solução, que representa o percentual da distância entre a solução encontrada e a solução ótima de um problema. Por meio do comportamento histórico dos dados, podemos analisar as características das instâncias e as configurações de parâmetros aplicadas em que se obteve o melhor *GAP*. Assim, através de um modelo de classificação, pode-se indicar de maneira fidedigna, a configuração de parâmetros que retorne um *GAP* ideal, próximo de zero.

[1] `https://projects.coin-or.org/Cbc`

3. Desenvolvimento

No processo de descoberta de conhecimento é realizada uma busca efetiva por conhecimentos no contexto da aplicação. Esse processo envolve a aplicação de algoritmos sobre os dados em busca de conhecimentos implícitos e úteis. Existem várias técnicas e algoritmos que podem ser utilizados no problema em questão, tais como redes neurais, algoritmos genéticos, modelos estatísticos e probabilísticos. Neste artigo foi utilizado a tarefa de classificação que consiste em descobrir uma função que mapeie um conjunto de registros em um conjunto de rótulos categóricos predefinidos, denominados classe.

No problema abordado, o conjunto de registros são as instâncias que possuem os atributos dos problemas de otimização combinatória e linear executados no CBC. Os atributos utilizados são relacionados aos principais componentes de um problema de PI:

- **variáveis**: contém atributos que apresentam o número total de variáveis presentes no problema, bem como o detalhamento da quantidade de variáveis binárias, inteiras e contínuas;
- **restrições**: contém atributos que contabilizam o número total de restrições do problema e os principais tipos de restrições (*set partition*, *set packing*, *set cover*, *cardinality*, *invariant knapsack*, *knapsack*, *binary flow conservation*, *integer flow conservation* e *continous flow conservation*);
- **matriz de incidência**: contém os valores mínimo e máximo dos coeficientes da matriz de incidência do problema.
- **conjunto de Parâmetros**: contém uma *string* que é o conjunto de parâmetros a serem utilizadas, passados como parâmetro para execução do problema no resolvedor CBC.

Além dos atributos das instâncias, temos a lista de alguns parâmetros que podem ser utilizados pelo resolvedor CBC. Abaixo segue a lista de parâmetros utilizados neste trabalho:

- **diveopt**: ativa ou desativa as heurísticas de mergulho (*diving*). No caso da ativação, decide em quais variáveis essas heurísticas serão aplicadas.
- **zero**: ativa ou desativa os cortes *Zero-Half* ($0/\frac{1}{2}$) [Caprara e Fischetti, 1996]. No caso da ativação, decide em quais ramos da árvore esses cortes serão gerados e inseridos.
- **strong**: número máximo de variáveis candidatas para aplicar *strong branching* [Achterberg et al., 2005]. Por padrão, o valor desse parâmetro é igual a 5.
- **trust**: número máximo de ramificações utilizando seleção de variáveis por *strong branching*. Por padrão, o valor desse parâmetro é igual a 5.
- **tunep**: parâmetro usado para controlar a etapa de pré-processamento. O pré-processamento inclui estratégias para alterar coeficientes e fixar os limites de algumas variáveis, diminuir o valor do lado direito de algumas restrições, etc.
- **proximity**: ativa ou desativa a heurística *Proximity Search* [Fischetti e Monaci, 2014].

A classe considerada neste trabalho representa o *GAP*, medida percentual da distância entre a solução encontrada e a solução ótima. Assim será identificada a melhor relação entre os atributos das instâncias e as configurações dos parâmetros que se aproxime do *GAP* ideal, próximo a zero.

Os algoritmos de classificação utilizados neste trabalho são implementados e executados pela ferramenta *Waikato Environment for Knowledge Analysis* (WEKA)[2], que é uma coleção de

[2] `http://www.cs.waikato.ac.nz/ml/weka/`

algoritmos de aprendizado de máquina e de pré processamento de dados. Esta inclui basicamente os algoritmos mais utilizados na literatura. Os algoritmos utilizados no trabalho são divididos em classes, sendo estes de árvore de classificação, aprendizagem preguiçosa (*lazy*), baseado em funções e os meta algoritmos. A Tabela 1 apresenta uma breve explicação sobre os algoritmos utilizados.

Nome	Função
RandomCommitee	Cria um conjunto de classificadores básicos aleatorizados. A previsão final é uma média linear das previsões geradas pelos classificadores de base.
RandomTree	Constrói uma árvore que considera um dado número de atributos aleatórios em cada nó.
RandomForest	Cria várias árvores de classificação A floresta escolhe a classificação com maior pontuação.
KStar	Utiliza o método do Vizinho mais próximo com função de distância generalizada.
J48	Utiliza o algoritmo de árvore de decisão c4.5.
Bagging	Meta algoritmo que melhora a estabilidade e a precisão para reduz a variância. Geralmente aplicado a métodos de árvore de decisão.
REPTree	Método de árvore de decisão rápido que usa poda reduzida de erros.
RandomSubSpace	Constrói um conjunto de classificadores de base com diferentes conjuntos de atributos.
ClassificationRegression	Executa classificação usando um método de regressão.
LMT	Constrói modelos de árvore de regressão logística.
LogitBoost	Executa regressão logística aditiva.
SimpleLogistic	Constrói modelos de regressão logística linear com seleção de atributos embutida.
MultiClassClassifier	Usa um classificador de duas classes para conjuntos de dados multiclasse.
Ibk	Utiliza um classificador baseado em K-vizinhos mais próximos.
LWL	Algoritmo geral para aprendizagem localmente ponderada.
SMO	Algoritmo de otimização mínima sequencial para a classificação de vetores de suporte.

Tabela 1: Tabela modificada de Witten et al. [2011]

3.1. Métricas de Avaliação

Métricas de avaliação são utilizadas para identificar os melhores algoritmos a serem aplicados sobre o problema em questão. Para tal avaliação, os algoritmos da Tabela 1, utilizados para a descoberta de conhecimento em base de dados, são comparados de acordo com métricas baseadas em erros e correlação entre os dados. As métricas utilizadas são: coeficiente de clusterização (CC), média do erro absoluto (MAE), raiz quadrada do quadrado do erro médio (RSME), erro relativo absoluto (RAE), raiz quadrada do erro relativo (RRSE) e o classificações corretas (CCI).

Formalmente o valor de θ é denotado como o valor verdadeiro de interesse, e $\hat{\theta}$ é denotado como valor estimado.

A métrica CC mostra a relação entre o valor verdadeiro e o valor estimado de uma classificação, com isso esse valor varia estritamente entre $-1\theta, \hat{\theta} \leq 1$. Quanto mais próximo os valores de θ e $\hat{\theta}$ de 0 a relação é basicamente inexistente e quanto mais próximo aos extremos a relação é muito forte, sendo, no caso negativo, uma relação muito forte inversa.

A métrica MAE (1), é uma das medidas mais comuns de erro de previsão. Essa medida não leva em conta se o erro foi sobrestimado ou subestimado, caracterizando-se por ser a média dos erros cometidos pelo modelo de previsão durante uma série de execuções. Para calcular, subtrai-se o valor da previsão ao valor verdadeiro em cada período de execução, o resultado deverá sempre ser positivo, sempre em módulo, soma-se e divide-se pelo número de valores que usado para obter a soma, representado a seguir formalmente:

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |\hat{\theta}_i - \theta_i| \tag{1}$$

A métrica RMSE (2), também é usada como uma medida do erro de previsão. É determinado a soma dos erros de previsão ao quadrado e dividindo pelo número de erros usado no cálculo. o RMSE pode ser expresso formalmente:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\hat{\theta}_i - \theta_i\right)^2} \tag{2}$$

A métrica RRSE (3), segundo Witten et al. [2011] refere-se a uma medida um pouco diferente, o cálculo do erro e feito em relação ao que teria sido um preditor simples utilizado. O preditor simples em questão é apenas a média dos valores reais dos dados, denotados por 1, assim o RRSE normaliza dividindo-se pelo erro quadrado total do indicador padrão, representado a seguir formalmente:

$$RRSE = \sqrt{\frac{\sum_{i=1}^{N}\left(\hat{\theta}_i - \theta_i\right)^2}{\sum_{i=1}^{N}\left(\overline{\theta} - \theta_i\right)^2}}, \text{sendo } \overline{\theta} \text{ a média dos valores de } \theta \tag{3}$$

A métrica RAE (4), segundo Witten et al. [2011] é apenas o erro absoluto total, sendo o mesmo tipo de normalização do RSSE, os error são normalizados pelo erros do preditor simples que prevê os valores médios, representado formalmente:

$$RAE = \frac{\sum_{i=1}^{N}|\hat{\theta}_i - \theta_i|}{\sum_{i=1}^{N}|\overline{\theta} - \theta_i|}, \text{sendo } \overline{\theta} \text{ a média dos valores de } \theta \tag{4}$$

A métrica de CCI a partir do resultado das classificações ele mostra em porcentagem a quantidade de instâncias que foram corretamente classificadas.

4. Experimentos Computacionais

As instâncias a serem utilizadas compreendem uma base de teste criada pelo Grupo de Otimização e Algoritmos (GOAL) da Universidade Federal de Ouro Preto. Essa base consiste de 199 problemas de Programação Inteira, incluindo instâncias da Mixed Integer Problem Library (MIPLIB) [Koch et al., 2011] e de problemas de escalonamento em geral.

Para obter uma base de dados de teste foram realizadas dezesseis execuções no CBC, para cada instância do conjunto. Essas execuções utilizam diferentes combinações de parâmetros deste resolvedor (ver Seção 3). Informações sobre cada uma das instâncias, além do detalhamento completo das execuções realizadas no CBC podem ser encontradas no sítio disponibilizado pelo GOAL [3].

A ferramenta *Waikato Environment for Knowledge Analysis (WEKA)* utilizada, foi projetada para que seja possível a experimentação rápida de métodos existentes em novos conjuntos de dados de maneira flexível Witten et al. [2011].

Sobre a base de testes, foram executados todos os algoritmos apresentados na Tabela 1, assim é possível avaliá-los em relação às métricas de avaliação apresentadas na subseção 3.1.

Analisando a Tabela 2 pode-se perceber, que para o problema em questão, os algoritmos que trabalham com fatores aleatórios, como *Random Commitee* e *Random Tree*, possuem desempenho melhor do que algoritmos que trabalham de forma sequencial, como por exemplo o SMO.

Os algoritmos *Random Commitee* e *Random Tree* obtiveram os resultados mais satisfatórios, ou seja, com os menores erros em relação às métricas de avaliação. Os mesmos possuem erros bem menores que os demais algoritmos, eles apresentam um coeficiente de clusterização(CC) próximo a 1, (ver Subseção 3.1), e classificam corretamente 98% da instâncias executadas.

A Tabela 2 apresenta os algoritmos ordenados de acordo com os menores erros, inciando com o MAE, RMSE, RAE e RRSE.

[3] `http://cbc.decom.ufop.br/`

Algoritmo	CC	MAE(%)	RMSE(%)	RAE(%)	RRSE(%)	CCI(%)
RandomCommitee	0.980	0.002	0.032	2.076	14.325	98.30
RandomTree	0.979	0.002	0.033	2.194	14.817	98.18
RandomForest	0.980	0.028	0.090	28.090	40.233	98.30
KStar	0.885	0.045	0.129	44.800	57.699	90.23
J48	0.578	0.056	0.167	55.814	74.730	64.45
Bagging	0.556	0.065	0.172	65.177	76.959	62.69
REPTree	0.461	0.069	0.186	69.390	83.325	54.55
RandomSubSpace	0.473	0.072	0.183	71.764	81.964	55.75
ClassificationRegression	0.415	0.080	0.193	79.577	86.385	51.22
LMT	0.313	0.080	0.205	80.279	91.605	43.59
LogitBoost	0.239	0.091	0.210	90.882	94.019	39.35
SimpleLogistic	0.071	0.096	0.218	95.688	97.670	29.40
MultiClassClassifier	0.064	0.096	0.218	95.815	97.757	28.89
Ibk	0.066	0.097	0.220	97.346	98.558	28.83
LWL	0.005	0.099	0.222	99.055	99.459	26.22
SMO	0.046	0.106	0.228	106.114	102.011	28.67

Tabela 2: Tabela de Comparação dos Algoritmos de Classificação

Na figura 1, é apresentado o modelo simplificado da árvore de classificação gerada pelo algoritmo *Random Tree*, apresentando somente os nós iniciais. O algoritmo que obteve o melhor desempenho, *Random Committee*, utiliza como classificador o *Random Tree* e encontra a média final das previsões geradas por esse classificador variando somente as bases.

Podemos observar que os nós principais correspondem aos atributos e quantidade de variáveis que o problema possui divididos em dois grupos, os que possuem valores menores que 552 e maiores ou iguais a 552. A partir do nó raiz da árvore expande-se uma sequência de sub árvores diferenciando os demais atributos em duas ou mais ramificações até que seja concluído a classificação. Os dois filhos diretos da raiz são *mflow* e *min*. O primeiro, *mflow*, é um atributo relacionado a um tipo de restrição que pode envolver variáveis inteiras e contínuas, cujo objetivo é garantir a conservação do fluxo em uma rede. O segundo, *min*, é o menor coeficiente da matriz do problema. Com isso, consegue-se perceber uma grande variação nas características da instâncias para o problema da recomendação de parâmetros, tornando impossível a percepção de algum conhecimento por meio desses atributos diretamente.

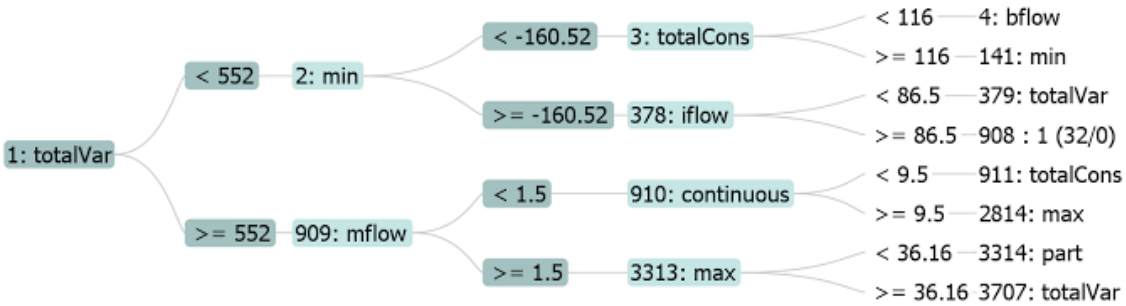

Figura 1: Modelo Simplificado da Árvore gerada pelo *Random Tree*

5. Conclusão e Trabalhos Futuros

Neste trabalho foi apresentada uma abordagem baseada em classificação para a recomendação de parâmetros a serem utilizados pelo *COIN-OR Branch and Cut* na resolução de problemas

de Programação Inteira. Nos experimentos realizados sobre uma base de dados diversificada foi possível obter uma taxa de acerto de 98% por meio da utilização de algoritmos baseados em árvores de decisão (*Random Commitee* e *Random Tree*). Com uma boa acurácia, é possível inferir qual conjunto de parâmetros associado ao CBC levaria à melhor solução possível para um dado problema.

Vale ressaltar que existe uma limitação quanto à recomendação dos parâmetros: foram selecionadas e utilizadas dezesseis combinações entre um vasto conjunto de parâmetros. Nesse sentido, destacam-se como trabalhos futuros a inclusão de novas combinações de parâmetros, a fim de ampliar a base de dados, refinar os resultados obtidos e obter uma boa taxa de acerto para novas instâncias. Além disso, pretende-se desenvolver uma aplicação que faça a recomendação dos parâmetros de forma independente da ferramenta WEKA, para que esse processo possa ser integrado diretamente com o uso do CBC. Por fim, pretende-se investigar a importância de cada atributo no processo de classificação, especialmente cada parâmetro e sua influência direta na produção de uma solução de qualidade.

Referências

Achterberg, T., Koch, T., e Martin, A. (2005). Branching rules revisited. *Operations Research Letters*, 33(1):42 – 54. ISSN 0167-6377. URL http://www.sciencedirect.com/science/article/pii/S0167637704000501.

Caprara, A. e Fischetti, M. (1996). {0, 1/2}-chvátal-gomory cuts. *Mathematical Programming*, 74 (3):221–235. ISSN 1436-4646. URL http://dx.doi.org/10.1007/BF02592196.

Fayyad, U., Piatetsky-Shapiro, G., e Smyth, P. (1996). The kdd process for extracting useful knowledge from volumes of data. *Commun. ACM*, 39(11):27–34. ISSN 0001-0782. URL http://doi.acm.org.ez28.periodicos.capes.gov.br/10.1145/240455.240464.

Fischetti, M. e Monaci, M. (2014). Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics*, 20(6):709–731. ISSN 1572-9397. URL http://dx.doi.org/10.1007/s10732-014-9266-x.

Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R. E., Danna, E., Gamrath, G., Gleixner, A. M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D. E., e Wolter, K. (2011). MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163.

Lougee-Heimer, R. (2003). The common optimization interface for operations research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, 47(1):57–66. ISSN 0018-8646.

Witten, I. H., Frank, E., e Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition. ISBN 0123748569, 9780123748560.