

# Construção de um protótipo de *framework* para otimização e seu uso na resolução do Problema de Roteamento de Veículos com Frota Heterogênea e Janelas de Tempo

**Tiago Araújo Neves, Marcone Jamilson Freitas Souza, Alexandre Xavier Martins**

Departamento de Computação, Universidade Federal de Ouro Preto,  
35.400-000, Ouro Preto, MG

E-mail: tiagoanz@yahoo.com.br, marcone@iceb.ufop.br, xmartins@uai.com.br,

## Resumo

Neste trabalho propomos um *framework*, denominado JFFO, para otimização através de metaheurísticas. Para sua validação, é construída uma aplicação para resolução do problema de roteamento de veículos com frota heterogênea e janelas de tempo. O *framework* torna possível a construção de aplicações para resolução de problemas em um curto período de tempo, além de tornar fácil a manutenção, extensão e modificação das aplicações. A aplicação gerada para validação utiliza as metaheurísticas *Simulated Annealing* e Busca Tabu, as quais integram o *framework*. Nessa aplicação, o procedimento *Simulated Annealing* é acionado com o objetivo de encontrar uma solução viável para o problema. A seguir, a solução resultante é refinada por um procedimento de Busca Tabu. A aplicação desenvolvida a partir do *framework* é comparada com o software de roteirização LOGWARE em termos da qualidade das soluções finais. Essa aplicação também é comparada em termos de manutenção, expansão e modificação, com uma outra que não faz uso do *framework*.

*Palavras-chave:* Framework, Roteamento de Veículos, Metaheurísticas.

## 1. Introdução

As técnicas de engenharia de software têm mostrado sua eficiência na melhoria dos softwares, principalmente no aumento do reuso e na interoperabilidade desses. Dentre essas técnicas, a construção de *framework* tem despertado interesse nas comunidades acadêmicas e de mercado, pois sua aplicação diminui o tempo de construção de softwares, aumenta sua confiabilidade e facilita a manutenção, expansão e modificação.

Destarte essas vantagens, pouco se tem relatado na literatura sobre experiências com a utilização dessa técnica na resolução de problemas de otimização combinatória [10]. Nesse sentido, este trabalho contribui com a proposição de um *framework* para resolver problemas de otimização através de metaheurísticas. Para validá-lo, é desenvolvida uma aplicação para resolver o Problema de Roteamento de Veículos com Frota Heterogênea e Janelas de Tempo (PRVFHJT).

O PRVFHJT é uma generalização do Problema de Roteamento de Veículos (PRV), e pode ser definido como segue: dado um conjunto de cidades (ou

consumidores), cada um com uma demanda  $q_i$  por um produto e com um conjunto de janelas de tempo (períodos de tempo em que o cliente pode ser visitado), e um depósito contendo veículos de diferentes capacidades, encontre as rotas dos veículos, minimizando os custos de transporte, atendendo a todas as demandas e não violando as janelas de tempo.

O interesse no PRV é devido à sua importância prática [1, 2, 3, 4] e também à sua grande dificuldade de solução. Como uma generalização do Problema do Caixeiro Viajante (PCV), o PRV pertence à classe de problemas NP-difíceis e um algoritmo em tempo polinomial para encontrar a solução ótima não é conhecido. Existem algoritmos exatos que raramente resolvem instâncias envolvendo mais que 50 cidades [18].

Devido ao limitado sucesso de métodos exatos, pesquisas têm sido desenvolvidas no sentido de desenvolver heurísticas capazes de encontrar boas soluções para instâncias de maior porte. Exemplos dessas heurísticas podem ser encontradas em [6, 16, 17, 18, 20, 21].

Já o interesse no PRVFHJT é devido à sua aplicação prática. Por ser uma variante do PRV, ele está mais próximo da realidade do que este último, uma vez que os clientes não abrem suas lojas em função das transportadoras e estas, em geral, não possuem frotas compostas apenas por veículos de mesma capacidade.

A aplicação desenvolvida a partir do *framework* proposto combina dois procedimentos metaheurísticos para resolver o PRVFHJT. Nessa aplicação, parte-se de uma solução inicial aleatória sobre a qual é aplicada a metaheurística *Simulated Annealing* para encontrar uma solução viável. Em seguida, a solução resultante é refinada por um procedimento de Busca Tabu (BT). Os resultados obtidos por essa aplicação são comparados com os produzidos pelo roteirizador LOGWARE. A aplicação desenvolvida também é comparada com uma outra desenvolvida de forma convencional, sem a utilização do *framework*.

## 2. Java Framework For Optimization

Um *framework* orientado a objetos é definido como um conjunto de classes que trabalham juntas, incorporando um padrão reutilizável para uma categoria de problemas. Ele dita a supra-estrutura da aplicação, descreve como as responsabilidades são divididas entre os diversos componentes e como esses componentes devem interagir entre si. O benefício de um *framework* é que o projetista de uma nova aplicação precisa se concentrar apenas em questões específicas do problema.

Decisões de projeto incorporadas pelo *framework* não precisam ser reexaminadas e nem o código provido pelo *framework* precisa ser reescrito.

As metaheurísticas são, em sua essência, independentes do problema. Elas são descritas como *templates* que são completamente desenvolvidas quando o problema alvo se encontra formalmente definido. Além disso, as metaheurísticas são intrinsecamente dependentes de estratégias e parâmetros, os quais são completamente especificados após um processo de sintonia fina, de acordo com a instância do problema a ser considerada. Os desenvolvedores de aplicações geralmente fazem muito esforço escrevendo e reescrevendo código, desviando a atenção do problema e dos métodos aplicados na resolução do mesmo, os quais deveriam ser o real foco da atenção desses desenvolvedores. Portanto, esses algoritmos se beneficiam fortemente de qualquer aspecto de engenharia de software que promova alta modularidade, reuso de software e interfaces de módulos bem definidas. Essas características levam a crer que a idéia de *framework* poderia ser aplicada, uma vez que apresenta boa parte, senão todas, das características que beneficiam as metaheurísticas.

Os *frameworks* têm algumas características de reuso de sistemas “caixa branca”, uma vez que um certo grau de conhecimento sobre sua estrutura é necessário para conseguir construir uma aplicação. Um grave problema que esse tipo de reuso traz é que, à medida que a complexidade da estrutura aumenta, maior deve ser o conhecimento sobre ela para se desenvolver uma aplicação. Além disso, o desenvolvimento de sistemas baseado em reuso de *frameworks* acarreta uma certa perda de desempenho.

Tendo em vista as idéias anteriores, desenvolveu-se a ferramenta *Java Framework For Optimization* (JFFO), a qual se destina a resolver problemas de otimização através de metaheurísticas, e tem sua arquitetura geral mostrada na Figura 1.

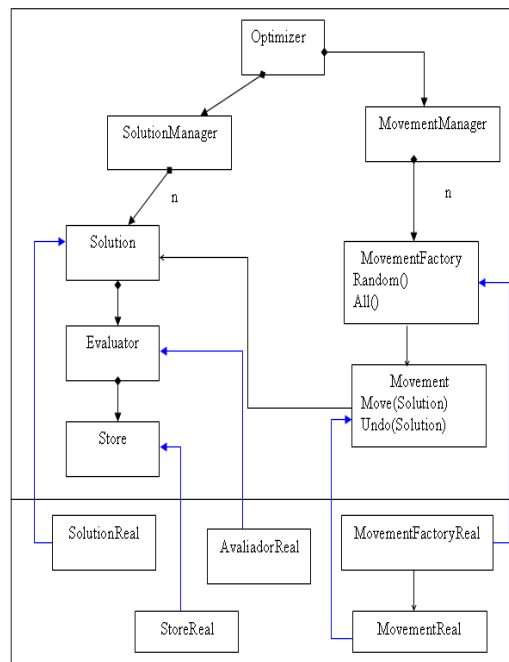


Figura 1 – Arquitetura do JFFO

As responsabilidades de cada classe são mostradas a seguir.

**Solution:** Responsável por manter uma representação de solução para um problema.

**SolutionManager:** Responsável por fazer o gerenciamento de uma ou mais soluções, viabilizando o uso simultâneo de mais de uma solução, tanto para heurísticas que trabalhem com um população de soluções quanto para heurísticas que precisem guardar soluções anteriormente examinadas.

**Movement:** Responsável por representar, executar e, eventualmente, desfazer um único movimento. Esta é a classe responsável por gerar os vizinhos.

**MovementFactory:** Responsável por construir os objetos para um determinado tipo de solução. Dentro da estrutura do *framework* ela trabalha como fábrica abstrata, sendo responsabilidade de suas subclasses construir os movimentos adequados. A presença desta classe é importante para possibilitar o uso de várias estruturas de vizinhança. Assim, cada *MovementFactory* representa uma estrutura de vizinhança.

**Optimizer:** Responsável pelo processo de otimização. Nesta versão, esta classe também faz o papel de mediador entre as classes *Solution* e *Movement*.

**Store:** Responsável por armazenar os dados necessários ao processo de otimização.

**Evaluator:** Responsável por avaliar as soluções durante o processo de otimização.

**MovementManager:** Responsável por gerenciar as diversas estruturas de vizinhança.

Com essa supra-estrutura em mãos, a criação de uma aplicação não é nada mais que uma extensão conveniente de cada classe para um determinado problema.

A ferramenta JFFO provê atualmente dois otimizadores: as metaheurísticas *Simulated Annealing* e Busca Tabu. Além disso, a ferramenta também provê estruturas de solução e estruturas de movimentação

gerais, caso o usuário deseje utilizar classes já testadas e corrigidas. Isso, contudo, não tira a liberdade do usuário de implementar suas próprias estruturas de solução e movimentação. Desde que ele respeite a hierarquia e as interfaces, ele poderá fazer suas estruturas interagirem com as demais classes do JFFO sem nenhum problema.

### 3. Formulação do Problema de Roteamento de Veículos

Seja  $G = (V, A)$  um grafo direto onde  $V = \{v_0, v_1, \dots, v_n\}$  é o conjunto de vértices e  $A = \{(v_i, v_j): i \neq j\}$  é o conjunto de arcos. O vértice  $v_0$  representa um depósito onde se encontra um conjunto de veículos, enquanto os vértices restantes correspondem às cidades ou consumidores. Cada consumidor  $v_i$  tem uma demanda não-negativa  $q_i$  (sendo que  $q_0 = 0$ ) e um conjunto de tuplas  $\{(t_o, t_c)\}$ . Cada tupla representa uma janela de tempo para o consumidor  $v_i$ , sendo  $t_o$  o tempo de abertura e  $t_c$  o tempo de fechamento de uma janela de tempo. A cada arco  $(v_i, v_j)$  está associada uma distância não-negativa  $c_{ij}$  que representa a distância entre os consumidores (nós). Dada uma velocidade média é possível calcular o tempo  $t_{ij}$  associado a cada arco.

O PRVFHJT consiste em determinar o conjunto de rotas que devem ser seguidas pelos veículos, minimizando os custos de transporte dado pelas distâncias e tempos e obedecendo às seguintes restrições:

- Cada rota começa e termina no depósito;
- Cada cidade de  $V \setminus \{v_0\}$  é visitada somente uma única vez por somente um veículo;
- A demanda total de cada rota não pode exceder a capacidade do veículo;
- Um cliente não pode ser atendido fora de sua janela de tempo.

### 4. Representação do PRVFHJT

No JFFO, uma solução genérica é composta por representantes, os quais podem ser bits, números ou letras, dependendo do problema com o qual se está trabalhando e da conveniência de usar uma ou outra representação.

Cada representante é uma tupla ('representa', 'local'). O campo 'representa' indica se esta tupla representa um veículo (0) ou um cliente (1), e o campo 'local' indica onde, na base de dados, encontra-se o representado. Há uma base de dados para clientes e outra para veículos. A representação da solução é um vetor de tuplas. Por exemplo, se há

3 cidades, 2 veículos e a solução  $s$  é  $\{(0-1), (1-3), (1-2), (0-2), (1-1)\}$ , então as rotas dos veículos, chamadas pétalas, são:  $\{(0-1), (1-3), (1-2)\}$  e  $\{(0-2), (1-1)\}$ . A primeira seqüência de tuplas  $\{(0-1), (1-3), (1-2)\}$  indica que o veículo que se encontra na primeira posição da base de dados de veículos irá atender aos clientes da terceira e segunda posições na base de dados de clientes. Na segunda rota, o veículo que se encontra na segunda posição na base de dados atenderá ao cliente que se encontra na primeira posição da base de dados de clientes.

### 5. Função de avaliação

Seja  $f_1(s)$  a distância total percorrida por todos os veículos da solução  $s$  e  $c_1$  o custo associado a cada unidade de distância. Seja  $f_2(s)$  a soma dos tempos de viagem de  $s$  e  $c_2$  o custo por unidade de tempo. Seja  $O_1(s)$  a sobrecarga dos veículos que seguem rotas acima de suas capacidades e  $O_2(s)$  a soma das violações às janelas de tempo. A função de avaliação é dada por  $f(s) = c_1 \times f_1(s) + c_2 \times f_2(s) + \alpha \times O_1(s) + \beta \times O_2(s)$ , sendo  $\alpha$  e  $\beta$  fatores de penalidade não-negativos.

### 6. Estrutura de vizinhança

Seja  $S$  o conjunto de soluções do PRVFHJT. A fim de derivar um algoritmo baseado em busca local para resolver esse problema, é necessário definir uma estrutura de vizinhança, isto é, uma função  $N$  à qual esteja associado um conjunto de soluções  $N(s)$ , com cada solução  $s \in S$  obtida por uma modificação parcial de  $s$ , chamada movimento. Neste trabalho são considerados dois tipos de movimentos, para definir uma única vizinhança  $N(s)$ . O primeiro movimento consiste em trocar duas tuplas da solução. Estas tuplas podem representar consumidores ou veículos. O segundo movimento consiste em mudar a posição de uma tupla mantendo-se a ordem das demais tuplas no vetor.

Assim, considerando a solução  $s$  da seção 4, um exemplo de vizinho de  $s$  gerado a partir do movimento de troca de tuplas é  $s' = \{(0-1), (1-3), (0-2), (1-2), (1-1)\}$ .

Um exemplo de vizinho da mesma solução  $s$  gerado pelo movimento de realocação de uma tupla no vetor é  $s' = \{(0-1), (1-2), (0-2), (1-3), (1-1)\}$ .

### 7. Método de solução

O método utilizado para resolver o problema combina as metaheurísticas *Simulated Annealing* (SA) e Busca Tabu (BT), as quais são usadas na seqüência para refinar uma solução gerada aleatoriamente. Descreve-se, a seguir, a adaptação dessas duas metodologias ao problema.

*Simulated Annealing* é uma classe de metaheurística proposta originalmente em [11], sendo uma técnica de busca local probabilística, que se fundamenta em uma analogia com a termodinâmica, ao simular o resfriamento de um conjunto de átomos aquecidos, operação conhecida como recozimento. Esta técnica começa sua busca a partir de uma solução inicial qualquer. O procedimento principal consiste em um *loop* que gera aleatoriamente, em cada iteração, um vizinho  $s'$  da solução corrente  $s$ .

A cada geração de um vizinho  $s'$  de  $s$ , é testada a variação  $\Delta$  do valor da função objetivo calculada conforme seção 5, isto é,  $\Delta = f(s') - f(s)$ . Se  $\Delta < 0$ , o método aceita a solução e  $s'$  passa a ser a nova solução corrente. Caso  $\Delta \geq 0$  a solução vizinha candidata também poderá ser aceita, mas neste caso, com uma probabilidade  $e^{-\Delta/T}$ , onde  $T$  é um parâmetro do método, chamado de temperatura e que regula a probabilidade de aceitação de soluções com custo pior.

A temperatura  $T$  assume, inicialmente, um valor elevado  $T_0$ . Após um número fixo de iterações (o qual representa o número de iterações necessárias para o sistema atingir o equilíbrio térmico em uma dada temperatura), a temperatura é gradativamente diminuída por uma razão de resfriamento  $\alpha$ , tal que  $T_n \leftarrow \alpha \times T_{n-1}$ , sendo  $0 < \alpha < 1$ . Com esse procedimento, dá-se, no início uma chance maior para escapar de mínimos locais e, à medida que  $T$  aproxima-se de zero, o algoritmo comporta-se como o método de descida, uma vez que diminui a probabilidade de se aceitar movimentos de piora ( $T \rightarrow 0 \Rightarrow e^{-\Delta/T} \rightarrow 0$ ).

O procedimento pára quando a temperatura chega a um valor próximo de zero e nenhuma solução que piore o valor da melhor solução é mais aceita, isto é, quando o sistema está estável.

Os parâmetros de controle do procedimento são a razão de resfriamento  $\alpha$ , o número de iterações para cada temperatura ( $SA_{max}$ ) e a temperatura inicial  $T_0$ .

A Busca Tabu [20] é um procedimento adaptativo dotado de uma estrutura de memória que aceita movimentos de piora para escapar de ótimos locais.

Mais especificamente, começando com uma solução inicial  $s_0$  gerada pelo procedimento SA, o

algoritmo BT explora, a cada iteração, a vizinhança  $N(s)$  da solução corrente  $s$ , considerando os dois tipos possíveis de movimento. O membro  $s'$  de  $N(s)$  com melhor valor nessa região segundo a função  $f(\cdot)$  torna-se a nova solução corrente mesmo que  $s'$  seja pior que  $s$ , isto é, que  $f(s') > f(s)$ .

O critério de escolha do melhor vizinho é utilizado para escapar de um ótimo local. Esta estratégia, entretanto, pode fazer com que o algoritmo cicle, isto é, que retorne a uma solução já gerada anteriormente. De forma a evitar que isto ocorra, existe uma lista tabu  $T$ , a qual é uma lista de movimentos proibidos. A lista tabu implementada é de tamanho dinâmico, isto é, varia no intervalo  $[|T|_{min}, |T|_{max}]$  e seu tamanho  $|T|$  é atualizado periodicamente. Esta lista contém os movimentos reversos aos últimos  $|T|$  movimentos realizados. Quando um novo movimento é adicionado à lista, o mais antigo é retirado. Como a lista tabu pode ser muito restritiva, isto é, movimentos tabu podem impedir o retorno a uma solução já visitada e também a soluções ainda não exploradas, um critério de aspiração é usado. No caso, é aceito um movimento tabu se ele gerar uma solução de qualidade superior à da melhor solução gerada até então.

O método pára se decorridos  $BT_{max}$  iterações sem melhora no valor da melhor solução.

## 8. Resultados Computacionais

O *framework* e a aplicação proposta foram implementados na linguagem Java. Para testar a aplicação, foram usadas as duas instâncias exemplo que acompanham o software LOGWARE e a instância 15 de [9]. Todos os experimentos foram realizados em um PC com processador Pentium IV, 2.8 GHz de *clock* H.T. e 512 MB de RAM. O método *Simulated Annealing* implementado é o apresentado em [19], com o número de iterações sem melhora em uma dada temperatura ( $SA_{max}$ ) fixado em 500 e a razão de resfriamento fixada em 0,97. A temperatura inicial do método é determinada por simulação, sendo aquela na qual 95% dos movimentos são aceitos. A cada iteração do método, um tipo de movimento (troca ou realocação) é selecionado aleatoriamente, sendo o vizinho da solução corrente gerado com esse movimento. O método de Busca Tabu implementado é o apresentado em [19], com os seguintes ingredientes adicionais: número máximo de iterações sem melhora fixado ( $BT_{max}$ ) em 210, lista tabu dinâmica com tamanho variando no intervalo [5, 10] e sendo modificado a cada 10 iterações. Na Busca Tabu, explora-

se a vizinhança completa da solução corrente considerando-se os dois tipos de movimentos.

Para cada instância foram executados 10 testes, cada qual partindo de sementes diferentes de números aleatórios. Na Tabela 1 são apresentados os resultados finais obtidos para as duas instâncias do LOGWARE, a primeira com janela de tempo (Com JT) e a segunda sem janela de tempo (Sem JT). São comparados os resultados obtidos pela aplicação desenvolvida a partir do JFFO, denominada AP, com os do LOGWARE. A coluna “Custo Médio AP” representa a média aritmética dos resultados das 10 execuções. A coluna “Custo LOGWARE” indica o valor da solução resultante pelo LOGWARE, o qual utiliza uma heurística convencional gulosa e, portanto, retorna sempre a mesma solução final, obtida de forma determinística. A coluna “% Melhora” indica o percentual de melhora obtida pela aplicação em relação ao LOGWARE.

Na Tabela 2 são comparados os tempos de execução. A colunas “Tempo Médio AP” representa a média dos tempos de execução (em segundos) das 10 execuções da aplicação.

Instância do Problema	Custo Médio AP	Custo LOGWARE	% Melhora
Sem JT	7953,1	8489	6,4
Com JT	4257,9	5095	16,5

Tabela 1 – Resultados computacionais (custo)

Instância do Problema	Tempo Médio AP (s)	Tempo Médio LOGWARE (s)
Sem JT	11,490	0,996
Com JT	45,456	0,996

Tabela 2 – Resultados computacionais (tempo)

Para a instância 15 de [9], que envolve 50 clientes sem janelas de tempo e 3 veículos de capacidades diferentes, o valor da melhor solução encontrada foi 2640, sendo que a média desses valores em 10 execuções do método foi 2710, obtidos em uma média de 127 segundos. Essa média encontra-se a 2,7% do melhor resultado da literatura, que é de 2640. O LOGWARE não foi capaz de resolver essa instância com o número de veículos relatado na literatura.

Do ponto de vista da engenharia de software,

também foram alcançados bons resultados, pois várias classes foram reutilizadas integralmente e muitas outras reutilizadas parcialmente através de especialização.

Para ilustrar o nível de reuso alcançado, o aplicativo proposto foi comparado com um aplicativo para resolução do mesmo problema, relatado em [15], construído sem o uso de técnicas de engenharia de software e que possui apenas o método *Simulated Annealing* implementado. A Tabela 3 compara os dois aplicativos com relação aos critérios manutenção, modificação, expansão e tamanho. A coluna “Outro Aplicativo” mostra os atributos de um aplicativo construído sem técnicas de engenharia de software.

Itens de avaliação	Aplicativo desenvolvido com o JFFO	Outro Aplicativo
Manutenção	Fácil	Difícil
Modificação	Fácil	Difícil
Expansão	Fácil	Difícil
Tamanho	115 K	137 K

Tabela 3 – Resultados de Engenharia de Software

A Tabela 3 mostra que a aplicação desenvolvida com o *framework* proposto possui menor tamanho, apesar de apresentar um método de otimização a mais, no caso, o método de Busca Tabu. Além disso, a aplicação proposta também se mostrou melhor nos quesitos manutenção, modificação e expansão, conforme a seguir se justifica.

A modularidade da aplicação proposta permite que ela seja expandida com a inclusão de novas classes. Permite também que ela sofra manutenção e modificação em partes independentes, sem que seja necessário propagar essas modificações por todas as classes do sistema.

O aplicativo desenvolvido sem o uso de técnicas de Engenharia de Software, por sua vez, por se tratar de um bloco monolítico, uma simples modificação, como a mudança do nome de uma variável, pode acarretar uma série de alterações em todo o código do aplicativo, dificultando sua manutenção, modificação e expansão.

## 9. Conclusões

Este trabalho propõe a construção de um *framework* para resolver problemas de otimização combinatória através de metaheurísticas, e valida seu uso com o desenvolvimento de uma aplicação para resolver o Problema de Roteamento de Veículos com Frota Heterogênea e Janelas de Tempo. A aplicação foi feita combinando-se as metaheurísticas *Simulated Annealing* e Busca Tabu.

Os resultados obtidos mostram que a aplicação desenvolvida a partir do *framework* produziu resultados

melhores que o software LOGWARE, comprovando a afirmação de [3], de que muitos softwares de mercado são baseados em metodologias ultrapassadas. Além disso, o aplicativo desenvolvido é de fácil manutenção, expansão e modificação, permitindo seu uso por meio de outras técnicas heurísticas e resolução de vários outros problemas combinatoriais correlatos.

## Referências

- [1] W. Bell, L. Dalberto, M. L. Fisher, A. Greenfield, R. Jaikumar, R. Mack, P. Prutzman, Improving distribution of industrial gases with an on-line computerized routing and scheduling systems, *Interfaces*, 13 (1983) 4-23.
- [2] G. Brown, G. Graves, Real-time dispatch of petroleum tank trucks, *Management Science*, 27 (1981) 19-32.
- [3] J. F. Cordeau, M. Gendreau, G. Laporte, J. Y. Potvin, F. SEMET, A guide to vehicle routing heuristics, *Journal of the Operational Research Society*, 53 (2002) 512-522.
- [5] M. L. Fisher, R. Greenfield, R. Jaikumar, J. Lester, A computerized vehicle routing application, *Interfaces*, 1 (1982) 45-52.
- [6] M. Gendreau, A. Hertz, G. Laporte, A tabu search heuristic for the vehicle routing problem, *Management Science*, 40 (1994) 1276-1290.
- [8] F. Glover, M. Laguna, Tabu Search, *Kluwer Academic Publishers*, Boston, (1997).
- [9] Golden, B.; Assad, A.; Levy, L.; Gheysens, F. (1984) The Fleet Size and Mix Vehicle Routing. *Computers and Operations Research*, v. 11, n. 1, p. 49-66.
- [10] M. Graccho, S. C. S. PORTO, TabOOBuilder: An Object-Oriented Framework for Building Tabu Search Applications. *Proceedings of the Third Metaheuristics International Conference*, (1999) 247-251.
- [11] S. Kirkpatrick, D. C. Gelott, M. P. Vecchi, Optimization by Simulated Annealing, *Science*, 220 (1983) 671-680.
- [12] G. Laporte, The Vehicle-Routing Problem – An Overview of exact and approximate algorithms, *European Journal of Operational Research*, 59 (1992) 345-358.
- [15] T. A. Neves, M. J. F. Souza, A. X. Martins, Resolução do Problema de Roteamento de Veículos com Frota Heterogênea e Janelas de Tempo. Relatório Técnico DECOM 01/2004. Departamento de Computação, Universidade Federal de Ouro Preto, (2004). Disponível em: <http://www.decom.ufop.br/prof/marcone/Orientacoes/PRVFHJTViaSimulatedAnnealing.pdf>.
- [16] I. H. Osman, Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem, *Annals of Operations Research*, 41 (1993) 421-451.
- [17] J. Renaud, F. F. Boctor, G. Laporte, An improved petal heuristic for the vehicle routing problem, *Journal of the Operational Research Society*, 47 (1996) 329-336.
- [18] J. Renaud, F. F. Boctor, A sweep-based algorithm for the fleet size and mix vehicle routing problem, *European Journal of Operational Research*, 140 (2002) 618-628.
- [19] Souza, M. J. F., Notas de aula de Inteligência Computacional para Otimização, Departamento de computação, Universidade Federal de Ouro Preto. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.pdf>
- [20] E. D. Taillard, Parallel iterative search methods for vehicle routing problems, *Networks*, 23 (1993) 661-673.
- [21] K. C. Tan, L. H. Lee, Q. L. Zhu, K. Ou, Heuristic Methods for Vehicle Routing Problem With Time Windows, *Artificial Intelligence in Engineering*, 15 (2001) 281-295.