DEPARTAMENTO DE COMPUTAÇÃO

D E C O M

Construção de um protótipo de *framework* para otimização e seu uso para a Resolução do problema de Roteamento de Veículos com Frota Heterogênea e Janelas de Tempo

Tiago Araújo Neves – 01.1.4126

Marcone Jamilson Freitas Souza
Marcelo de Almeida Maia
Alexandre Xavier Martins

Relatório Técnico DECOM 01/2005

U F O P

UNIVERSIDADE FEDERAL DE OURO PRETO

Tiago Araújo Neves

CONSTRUÇÃO DE UM PROTÓTIPO DE *FRAMEWORK* PARA OTIMIZAÇÃO E SEU USO PARA A RESOLUÇÃO DO PROBLEMA DE ROTEAMENTO DE VEÍCULOS COM FROTA HETEROGÊNEA E JANELAS DE TEMPO

MONOGRAFIA SUBMETIDA AO DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO DA UNIVERSIDADE FEDERAL DE OURO PRETO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Prof. Dr. Marcone Jamilson Freitas Souza
Doutor pela COPPE/Universidade Federal do Rio de Janeiro
Departamento de Ciência da Computação, UFOP

Prof. Alexandre Xavier Martins
Mestrando pela Universidade Federal de Minas Gerais
Departamento de Ciência da Computação, UFOP

Prof. Dr. Marcelo de Almeida Maia
Doutor pela UFMG
Departamento de Ciência da Computação, UFOP

OURO PRETO, MG – BRASIL
JANEIRO DE 2005

Resumo

Este trabalho tem seu enfoque no desenvolvimento e teste de um *framework* para fins de otimização, que é uma área onde a engenharia de software é extremamente aplicável e na construção de uma aplicação, utilizando o *framework* proposto, para a resolução do Problema de Roteamento de Veículos com Frota Heterogênea e Janelas de Tempo (PRVFHJT) de uma empresa de transporte. Este problema consiste na construção de rotas que atendam a conjunto de clientes, de tal forma que as rotas sejam executadas com o menor custo possível.

O número elevado de clientes e de suas possíveis ordens de visitação para constituir as rotas, associado com a necessidade de respeitar as regras operacionais das empresas e de minimizar custos, torna altamente complexa a atividade de construção de rotas. Na literatura o PRVFHJT é considerado NP-difícil, o que dificulta, ou até mesmo impossibilita a resolução de casos reais por métodos de programação matemática, dito exatos. Desta forma, o problema é normalmente abordado por técnicas heurísticas.

Nesse trabalho, o PRVFHJT foi tratado levando-se em consideração a metaheurística *Simulated Annealing* (SA) e *Tabu Search* (TS), utilizando como movimentos de vizinhança a permutação de clientes e veículos na solução: podendo as trocas ocorrer entre clientes da mesma rota (intra-rota), entre clientes de rotas diferentes(inter-rota), entre um veículo e um cliente (diminuição/aumento de rota) e entre veículos (troca de rotas) bem como movimentos de realocação, onde apenas um único cliente ou veículo é deslocado, mantendo os demais clientes e veículos como estavam.

A função de avaliação utilizada pelos métodos visa a eliminação de inviabilidades, tais como a sobrecarga de um veículo, o não atendimento de um cliente e outras, e ainda visa a redução dos custos operacionais, que consistem em minimizar a distância das rotas percorridas e redução dos custos de espera.

O *framework* e a aplicação para a instância do problema desenvolvidos foram implementados utilizando-se a linguagem de programação Java, com a ferramenta TextPad.

Para simplificação do problema, a interface com o usuário não foi desenvolvida, focando-se assim, apenas na resolução do problema. Com o intuito de validar o sistema implementado, testaram-se os resultados obtidos por este, com os produzidos por um software de roteirização de mercado, o LogWare.

Agradecimentos

A Deus, pela saúde e pela ajuda que mais ninguém pode dar.

À minha família, em especial à minha avó Tereza, aos meus pais Willes e Consolação e a meu tio Afonso pelo apoio e incentivo durante a realização do curso de Ciência da Computação.

Aos professores Marcone Jamilson Freitas Souza e Marcelo de Almeida Maia, pela orientação para a realização deste trabalho, e principalmente pelo companheirismo e amizade.

Aos professores Alexandre Xavier Martins e Lucília Camarão Figueredo pelo apoio e confiança.

Aos grandes amigos que fiz em Ouro Preto. Rodrigo de Oliveira Furtado, Raphael Eustáquio Vilella, André Junqueira dos Santos, Deângelo Carvalho de Oliveira e Rodrigo Geraldo Ribeiro.

Aos amigos de outras cidades, Anderson Ferreira, Rafael Ferreira, Haroldo Gomes, Danilo Gomes, Klaus Lana, Renato Silva, Mateus Laurenço, Carlos Dias e Alexandre Santos, que sempre que precisei, estenderam a mão para me ajudar.

Às Amigas Marina, Greyce e Tathielle.

A todos os demais, que de alguma forma contribuíram para a realização deste trabalho.

Índice

Lista de Figuras

Lista de Tabelas

1 Introdução

O PRVFHJT é um problema derivado da classe dos problemas de roteamento de Veículos, o qual será descrito a seguir juntamente com seus derivados.

O PRV ou problema de roteamento de veículos consiste em construir um conjunto de rotas de veículos, cada uma começando e terminando no depósito. Os veículos possuem todos a mesma capacidade, e o objetivo da resolução do problema é reduzir a distância total percorrida.

O Problema de Roteamento de Veículos com Frota Heterogênea (PRVFH) tem as mesmas características do PRV, exceto que neste problema os veículos possuem capacidades diferentes [4]. Neste ponto a literatura divide o PRVFH em duas "classes" diferentes: a mais comum, na qual o conjunto de veículos possui uma quantidade ilimitada de cada tipo de veículos. Aqui, o objetivo foca-se em encontrar a melhor frota e otimizar as rotas. A segunda classe, que possui um número fixo de cada tipo de veículo, tem por objetivo otimizar o uso de uma frota (com número de veículos bem definidos) no processo de roteamento.

De fato a segunda classe na classificação de Gendreau *et al*. [4] é a que mais se aproxima da realidade, uma vez que as companhias de transporte na maioria das vezes já possuem uma frota. Um uso prático da primeira classificação seria o caso de uma companhia que ainda está adquirindo sua frota ou uma companhia que esteja disposta a mudar a sua frota, o que, em ambos os casos, requer uma grande quantia de capital, o que nem sempre é fácil para empresas que estão começando suas atividades.

Para aproximar mais o problema tratado da realidade, deve-se levar em conta que os clientes não podem ser atendidos a qualquer horário. Em outras palavras não é de se esperar que uma loja esteja disposta a receber sua mercadoria fora do seu horário de funcionamento. Para isso é necessário modelar os períodos de tempo nos quais os clientes podem ser atendidos. Tal artefato é chamado de **janela de tempo** e é modelado no formato de um intervalo (inicio e fim), dentro do qual um cliente pode ser atendido. Cada cliente pode possuir várias janelas de tempo. O PRVFHJT trata das janelas de tempo e dos outros aspectos mencionados anteriormente e também está sujeito à divisão de Gendreau *et al*. [4].

A instância de problema abordada neste trabalho será um pouco mais complexa do que o PRVFHJT, pois levará em conta a possibilidade de compartimentação dos veículos, que em muitos casos é necessária para transporte de produtos diferentes, como é o caso de veículos que possuem uma câmara refrigerada e outra parte não refrigerada, por exemplo.

Outro objetivo do projeto é a proposição, construção e teste de um protótipo de *framework* para otimização, visto que o uso de técnicas de engenharia de software pode melhorar a velocidade de construção de softwares na área de otimização e também a qualidade destes.

Este *framework*, batizado de *Java Framework For Optimization* (JFFO), tem como objetivo fornecer uma super-estrutura para que metaheurísticas, como *Simulated Annealing* e *Tabu Search,* possam ser utilizadas sem necessidade de reconstrução destes algoritmos, sendo necessário apenas a configuração de parâmetros. Também é provido um conjunto de modelos de soluções e de estruturas de movimentação, para que o usuário do JFFO tenha maior facilidade para construir uma aplicação. A liberdade porém, não é afetada, uma vez que o usuário tem liberdade para criar seus próprios modelos de solução e estruturas de movimentação para os modelos por ele definidos, desde que a integridade da estrutura do *framework* não seja violada.

1.1 Estrutura do trabalho

O presente trabalho está dividido em seis capítulos, incluindo esta introdução.

No capítulo II faz-se uma breve revisão bibliográfica sobre as técnicas de construção de *frameworks* para fins de otimização e sobre as técnicas de resolução do VRPFHJT e descrevem-se os métodos heurísticos considerados neste trabalho para resolvê-lo.

O capítulo III descreve o PRVFHJT abordado, bem como o modelo heurístico de otimização desenvolvido para resolvê-lo.

No capítulo IV são mostrados detalhes do sistema computacional implementado.

O capítulo V mostra os resultados obtidos pela aplicação do sistema.

No capítulo VI o trabalho é concluído com a análise dos resultados obtidos e sugestões para trabalhos futuros.

2 Revisão Bibliográfica

Nos últimos 40 anos o problema de roteamento de veículos e seus problemas derivados tem sido massivamente estudados. Destaque é dado para alguns métodos heurísticos e metaheurísticos.

Golden [2] faz uma descrição detalhada de vários métodos heurísticos, dentre os quais se pode citar a clássica heurística de Clark & Wright e o procedimento Giant Tour.

Nos últimos 10 anos os esforços foram voltados para o desenvolvimento das metaheurísticas, utilizando dois princípios básicos: busca local, objetivando a intensificação da busca de melhores soluções em um dado local, e busca na população global, de forma a promover a diversificação e permitir a combinação entre as soluções geradas.[7]

Gendreau, Laport e Potvin [8] descrevem e comparam uma série de métodos metaheurísticos como, por exemplo, o *Taburoute* e o *Simulated Annealing* de Osman.

O problema de roteamento de veículos e algumas de suas variantes são descritos a seguir.

2.1 O Problema de Roeteamento de Veículos - PRV

O Problema de Roteamento de Veículos (PRV) é um dos problemas mais estudados dentro da área de otimização combinatória [9].

A noção de roteamento de veículos já é bastante comum no cotidiano das pessoas. Desde a distribuição de gás à coleta de lixo, da entrega de encomendas normais às de grande porte e urgência, o problema básico de definição de rotas para que os veículos de transporte possam entregar suas encomendas se faz presente.

Um problema típico de roteamento de veículos (PRV) pode ser descrito, conforme Bräysy [13], como o problema de definição do menor custo das rotas de um depósito a um conjunto de clientes distribuídos em um plano geográfico. As rotas devem ser definidas para que cada cliente seja visitado somente uma única vez por exatamente um único veículo, começando e terminando no depósito, de forma a não exceder a capacidade do veículo. Todas as demandas dos clientes devem ser satisfeitas.

Uma revisão sobre diversos métodos de solução para o PRV, bem como a apresentação de diferentes variantes do problema é encontrada em CORDEAU *et al.* [9]. Neste trabalho os procedimentos de solução existentes na literatura são classificados em quatro categorias, a saber, precisão, velocidade, simplicidade e flexibilidade.

Uma heurística melhorada baseada no método *Simulated Annealing* é, por sua vez, apresentada em Breedam [14], que compara os resultados a outras heurísticas da literatura. O método proposto consiste de duas fases. Na primeira fase, procura-se minimizar as distâncias e tempos de deslocamento de cada rota pela reordenação da seqüência dos clientes a serem atendidos. Na segunda fase procura-se realocar os clientes entre duas rotas de uma solução viável.

Diversas são as variações do PRV básico, na maioria das vezes apenas compostos por um maior número de restrições, seja de tempo, de distância, de composição de frota, de limitação de frota etc. As variantes do PRV mais comuns são aquelas nas quais a frota é heterogênea e aquelas nas quais existem restrições com respeito a janelas de tempo.

2.2 Problema de Roteamento de Veículos com Frota Heterogênea

Este problema consiste no PRV básico com a restrição de que a frota é composta por tipos diferentes de veículos de transporte. O que diferencia um tipo de veículo de outro é a sua capacidade de transporte. A frota de veículos de cada tipo é normalmente limitada.

Em Gendreau *et al.* [4], tal problema foi resolvido utilizando o método de Busca Tabu. Os autores adaptaram a heurística denominada GENIUS (GENI: *Generalized Insertion* e US: *Unstringing and Stringing*), para resolver o Problema do Caixeiro Viajante para a construção da solução inicial do PRV e uma heurística baseada no método das economias, aliado a um método de memória adaptativa, para refinar a solução construída.

Uma outra abordagem para o problema, apresentado por Renaud & Boctor [3], utiliza um método baseado na heurística de varredura (*Sweep Based Algorithm*), auxiliado por cinco outros procedimentos denominados: *Order* (ordena os clientes em ordem crescente de distância a partir do vértice de referência), *1-petal* (seleciona uma rota, minimiza a distância desta e aloca um veículo), *2-petal* (seleciona 2 rotas, minimiza suas distâncias e aloca 2 veículos às mesmas), *petals selection* (seleciona um conjunto composto por uma configuração *1-petal* e uma configuração *2-petal*, combinando os clientes) e *improve* (procedimento de melhora das rotas). Os resultados apresentados são comparados a outros da literatura.

Uma abordagem para o PRV com frota heterogênea e janela de tempo é apresentada por LIU *et al.* [15]. Os autores utilizaram uma versão modificada do método das economias para inserção dos clientes nas rotas, onde um fator *N* (fator de decremento) determina a variação dos resultados através da sua multiplicação pelo número consecutivo de rotas construídas com pelo menos dois clientes. Utiliza-se desse recurso pelo fato de que, segundo os autores, muitas construções consecutivas compostas por dois ou mais clientes resultam numa solução de péssima qualidade no atendimento a restrições de janelas de tempo.

O PRV com restrições de janela de tempo, como o apresentado, é uma das variações mais implementadas na literatura e é apresentada a seguir.

2.3 O Problema de Roeteamento de Veículos com Janelas de Tempo

Uma das variações do PRV mais implementadas, o PRV com janelas de tempo, já possui diversas contribuições na literatura.

Também conhecido como Problema de Roteamento de Veículos com Janela de Tempo (PRVJT), ou na literatura inglesa como VRPTW (*Vehicle Routing Problem with Time Windows*), o problema consiste na adição de restrições de intervalo de tempo de atendimento ao problema básico do PRV.

Mais precisamente, dado um tempo de serviço s_i ,cada janela de tempo é definida como $[a_i,b_i]$, onde a_i é o horário mais cedo e b_i o horário mais tarde que se pode começar o atendimento. Caso o veículo chegue ao cliente antes do horário definido por a_i, este deve esperar. Caso chegue após b_i, este não poderá ser atendido.

Mine [7] ainda cita diversos métodos que já foram utilizados na resolução do problema de roteamento de veículos com janela de tempo descritos na literatura. Dentre estes estão Algoritmos Genéticos, Método Reativo de Busca em Vizinhança Variável e GRASP.

2.4 Métodos utilizados

2.4.1 Método metaheurístico *Simulated Annealing*

Simulated Annealing é uma classe de metaheurística proposta originalmente por Kirkpatrick et al. (1983), 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 Δ do valor da função objetivo, 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 α, tal que $T_n \leftarrow \alpha * T_{n-1}$, 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 α, o número de iterações para cada temperatura (*SAmax*) e a temperatura inicial T_0. A Figura 2.4.1.1 apresenta o algoritmo *Simulated Annealing* básico.

O *Simulated Annealing* foi escolhido para ser uma das metaheurística deste trabalho por se tratar de uma das mais eficientes metaheurísticas citadas na literatura tanto em qualidade de soluções apresentadas quanto pela velocidade de execução.

```
Procedimento S.A. (f(.), N(.), α, SAmax, T0, s)
1.  s* ← s;                  {Melhor solução obtida até então}
2.  IterT ← 0;               {Número de iterações na temperatura T}
3.  T ← T0;                  {Temperatura corrente}
4.  enquanto (T > 0) faça
5.     enquanto (IterT < SAmax) faça
6.        IterT ← IterT + 1;
7.        Gere um vizinho qualquer s' ∈ N(s);
8.        Δ = f(s') – f(s);
9.        se (Δ < 0)
10.          então s ← s';
11.                se (f(s') < f(s*)) então s* ← s';
12.          senão Tome x ∈ [0,1];
13.                se (x < e^(-Δ/T)) então s ← s';
14.       fim-se;
15.    fim-enquanto;
16.    T ← α × T;
17.    IterT ← 0;
18. fim-enquanto;
19. s ← s*;
20. Retorne s;
```

Figura 2.4.1.1 – Algoritmo *Simulated Annealing*

2.4.2 *Método metaheurístico Busca Tabu*

A Busca Tabu [12] é um procedimento adaptativo dotado de uma estrutura de memória e que aceita movimentos de piora para escapar de ótimos locais.

Mais especificamente, começando com uma solução inicial $s0$, um algoritmo BT explora, a cada iteração, um subconjunto V da vizinhança $N(s)$ da solução corrente s. O membro $s0$ de V com melhor valor nessa região segundo a função $f(:)$ torna-se a nova solução corrente mesmo que $s0$ seja pior que s, isto é, que $f(s0) > f(s)$ para um problema de minimização.

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 clássica contém os movimentos reversos aos últimos $|T|$ movimentos realizados (onde $|T|$ é um parâmetro do método) e funciona como uma fila de tamanho fixo, isto é, quando um novo movimento é adicionado à lista, o mais antigo é retirado.

Assim, na exploração do subconjunto V da vizinhança $N(s)$ da solução corrente s, ficam excluídos da busca os vizinhos $s0$ que são obtidos de s por movimentos m que constam na lista tabu.

A lista tabu se, por um lado, reduz o risco de ciclagem (uma vez que ela garante o não retorno, por jTj iterações, a uma solução já visitada anteriormente); por outro, também pode proibir movimentos para soluções que ainda não foram visitadas. Assim, existe também uma função de aspiração, que é um mecanismo que retira, sob certas circunstâncias, o status tabu de um movimento. Mais precisamente, para cada possível valor v da função objetivo existe um nível de aspiração $A(v)$: uma solução s' em V pode ser gerada se $f(s') \leq A(f(s))$, mesmo que o movimento m esteja na lista tabu. A função de aspiração A é tal que, para cada valor v da função objetivo, retorna outro valor $A(v)$, que representa o valor que o algoritmo aspira ao chegar de v. Considerando uma função objetivo de valores inteiros, um exemplo simples de aplicação desta idéia é considerar $A(f(s)) = f(s^*) - 1$ onde s^* é a melhor solução encontrada até então. Neste caso, aceita-se um movimento tabu somente se ele conduzir a um vizinho melhor que s^*.

Duas regras são normalmente utilizadas de forma a interromper o procedimento. Pela primeira, pára-se quando é atingido um certo número máximo de iterações sem melhora no valor da melhor solução. Pela segunda, quando o valor da melhor solução chega a um limite inferior conhecido (ou próximo dele). Esse segundo critério evita a execução desnecessária do algoritmo quando uma solução ótima é encontrada ou quando uma solução é julgada suficientemente boa.

Os parâmetros principais de controle do método de Busca Tabu são a cardinalidade $|T|$ da lista tabu, a função de aspiração A, a cardinalidade do conjunto V de soluções vizinhas testadas em cada iteração e $BTmax$, o número máximo de iterações sem melhora no valor da melhor solução.

Neste procedimento $fmin$ é o valor mínimo conhecido da função f, informação essa que em alguns casos está disponível.

É comum em métodos de Busca Tabu incluir estratégias de intensificação, as quais têm por objetivo concentrar a pesquisa em determinadas regiões consideradas promissoras.

Uma estratégia típica é retornar a soluções já visitadas para explorar sua vizinhança de forma mais efetiva. Outra estratégia consiste em incorporar atributos das melhores soluções já encontradas durante o progresso da pesquisa e estimular componentes dessas soluções a tornar parte da solução corrente. Nesse caso, são consideradas livres no procedimento de busca local apenas as componentes não associadas às boas soluções, permanecendo as demais componentes fixas. Um critério de término, tal como um número fixo de iterações, é utilizado para encerrar o período de intensificação.

Métodos baseados em Busca Tabu incluem, também, estratégias de diversificação. O objetivo dessas estratégias, que tipicamente utilizam uma memória de longo prazo, é redirecionar a pesquisa para regiões ainda não suficientemente exploradas do espaço de soluções.

Estas estratégias procuram, ao contrário das estratégias de intensificação, gerar soluções que têm atributos significativamente diferentes daqueles encontrados nas melhores soluções obtidas. A diversificação, em geral, é utilizada somente em determinadas situações, como, por exemplo, quando dada uma solução s, não existem movimentos m de melhora para ela, indicando que o algoritmo já exauriu a análise naquela região. Para escapar dessa região, a idéia é estabelecer uma penalidade $w(s;m)$ para uso desses movimentos. Um número fixo de iterações sem melhora no valor da solução ótima corrente é, em geral, utilizado para acionar essas estratégias.

Métodos de Busca Tabu incluem, também, listas tabu dinâmicas, muitas das quais atualizadas de acordo com o progresso da pesquisa. A grande vantagem de se usar uma lista tabu de tamanho dinâmico é que se minimiza a possibilidade de ocorrência de ciclagem.

Procedimento *B.T.* (f(.), N(.),A(.), |V|,f_{min}, |T|, BTmax, *s*)
1. *s** ← *s*; {Melhor solução obtida até então}
2. *IterT* ← 0; {contador do número de iterações }
3. *MelhorIter* ← 0 {Iteração mais recente que forneceu s*}
4. *T* ←Ø; {Lista tabu}
5. Inicialize a função de aspiração A;
6. enquanto (*f(s)* > fmin e I*ter –MelhorIter<BTmax*) faça
7. Iter ← Iter + 1;
8. Seja *s'* ← *s* ⊕ *m* o melhor elemento de *V ½ N*(*s*) tal que o movimento *m* não seja tabu (*m 62 T*) ou *s0* atenda a condição de aspiração (*f*(*s'*) < *A*(*f*(*s*)));
9. *T* ← *T* - {movimento mais antigo} + {movimento que gerou *s'*};
10. Atualize a função de aspiração *A*;
11. *s* ← *s'*;
12. se (*f(s')* < *f(s*)*) então
13. *s** ← *s'*;
14. MelhorIter ← Iter;
15. fim-se;
16. fim-enquanto;
17. *s* ← *s**;
18. Retorne s;

Fim *B.T.*;

Figura 2.4.2.1 – Algoritmo Busca Tabu

2.4.3 Método dos Quadrados Mínimos

Muitos dos problemas abordados na literatura relativos a roteamento de veículos trabalham com coordenadas cartesianas, o que implica que em muitos casos as distâncias utilizadas no cálculo dos custos são distâncias euclidianas. Isto é um fator que empobrece o problema no que diz respeito à realidade, afinal, as distâncias euclidianas, que são as distâncias em linha reta de um ponto ao outro, têm uma probabilidade mínima de serem as distâncias reais entre estes mesmos pontos, uma vez que não são considerados as curvas das estradas e nem os desníveis de altitude, que são variáveis presentes no mundo real.

Usar as distâncias reais como parâmetro para o problema é a solução ideal, mas nem sempre possível. Quando o numero de clientes torna-se grande, o conjunto de distâncias de entrada torna-se muito grande uma vez que a distância entre todos os vértices (Clientes e deposito) é necessária. Supondo que existam "n" vértices, o número de arestas necessárias é "n^2", o que dificulta a entrada de dados quando o numero de vértices se torna relativamente grande pois o volume de entrada torna-se muito grande e nem sempre é possível saber a distância real entre todos os vértices.

Uma maneira interessante de abordar o problema é utilizar mecanismos que levem em conta os dois conceitos, utilizando as distâncias reais conhecidas quando possível e as distâncias euclidianas quando não for possível o uso das distâncias reais.

Um método interessante neste aspecto é o método dos Quadrados Mínimos. Este método utiliza um conjunto de distâncias reais para gerar dados estatísticos, estes dados por sua vez são utilizados, juntamente com as distâncias euclidianas, para calcular uma estimativa das distâncias reais entre os vértices. Tornando assim, o cálculo das distâncias entre os pares de vértices que não foram mencionados no conjunto de distâncias reais de entrada, mais adequado ao modelo real do problema. A apresentação do método dos Quadrados Mínimos feita a seguir foi baseada na descrição de Souza [5].

Mostremos, inicialmente, como ajustar um conjunto de pontos a uma reta $y = a + bx$, onde a e b são parâmetros a serem determinados.

Neste caso, estamos interessados em minimizar a distância de cada ponto $(x_i; y_i)$ da tabela à cada ponto $(x_i; a + bx_i)$ da reta, conforme ilustra a Figura 2.4.3.1.

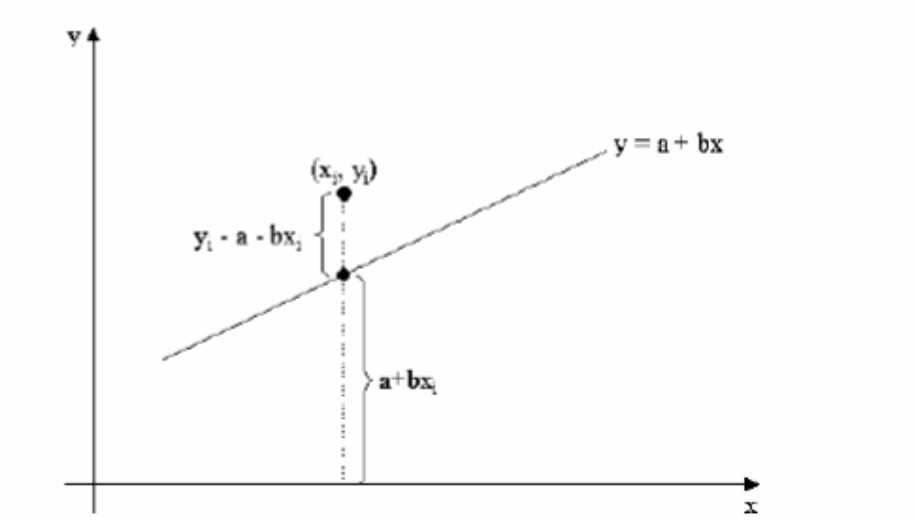

Figura 2.4.3. 1 Distância de um ponto (xi;yi) à reta y=a + bx

A distância entre esses pontos é $y_i – a - bx_i$ e a soma dos quadrados dessas distâncias é:

$$q = \sum_{i=1}^{n} (y_i - a - bx_i)^2 \qquad (2.1)$$

Os candidatos a ponto de mínimo da função 2.1 são aqueles para os quais são nulos as derivadas parciais de q em relação a cada um de seus parâmetros, isto é:

$$\frac{\partial q}{\partial a} = -2 \sum_{i=1}^{n} (y_i - a - bx_i) = 0 \qquad (2.2)$$

$$\frac{\partial q}{\partial b} = -2 \sum_{i=1}^{n} x_i (y_i - a - bx_i) = 0 \qquad (2.3)$$

Tendo em vista que:

$$\begin{aligned}\sum_{i=1}^{n}(y_i - a - bx_i) &= \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} a - \sum_{i=1}^{n} bx_i \\ &= \sum_{i=1}^{n} y_i - na - \left(\sum_{i=1}^{n} x_i\right) b\end{aligned}$$

e que:

$$\sum_{i=1}^{n} x_i(y_i - a - bx_i) = \sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right) a - \left(\sum_{i=1}^{n} x_i^2\right) b$$

Obtemos o seguinte conjunto de equações denominado "equações normais" do problema, cujas incógnitas são os parâmetros "a" e "b" da equação y = bx + a.

$$\begin{cases} na + \left(\sum_{i=1}^{n} x_i\right) b = \sum_{i=1}^{n} y_i \\ \left(\sum_{i=1}^{n} x_i\right) a + \left(\sum_{i=1}^{n} x_i^2\right) b = \sum_{i=1}^{n} x_i y_i \end{cases}$$

Tornando "x_i" a distância euclidiana e "y_i" como a distância real de um dado subconjunto de pontos, este método pode ser aplicado para estimar as distâncias reais de todo o conjunto de pontos de problemas de roteamento. Para isso, basta aplicar a fórmula $y_i = bx_i + a$ para estimar todas as distâncias, inclusive aquelas para as quais se conhece a distância real. Na literatura é comum estimar a distância real através da fórmula: d_real = *k**d_euclidiana, sendo *k* um fator de correção que depende da região sob análise. Para obter esse fator *k* basta dividir a média das distâncias estimadas (pela aplicação do método dos quadrados mínimos) pela média das distâncias euclidianas.

2.5 A engenharia de software em Otimização

Por lidar com problemas de natureza NP-Difícil, na área de otimização o tempo e o desempenho dos softawares desenvolvido são fatores críticos, forçando, muitas vezes, os desenvolvedores a criarem artifícios e fazerem uso de técnicas não recomendadas pela boa pratica de programação para conseguir melhor desempenho.

Contudo, quando o tempo não é um fator critico, o uso de técnicas de engenharia de software pode ajudar na construção, adaptação e manutenção de um software.

Segundo Graccho e Porto [11] o uso de técnicas de orientação a objetos, por exemplo, trazem uma série de benefícios como: (i) promover e facilitar a reusabilidade de software; (ii) facilitar a interoperabilidade; (iii) produzir soluções que se assemelham com o problema original; e (iv) resultam em softwares que são de facil manutenção e que são facilmente modificáveis e extensíveis. Além disso, uma classe define concisamente um componente de software, permitindo que esta seja analizada, descrita e testada isoladamente.

Durante os últimos anos tem crescido o interesse em descrever a maneira como coleções de classes funcionam juntas na solução de problemas genéricos, este interesse pode

ser claramente ilustrado por uma idéia que se tornou muito popular nos últimos anos. Os chamados *frameworks*.

Um *framework* orientado a objetos é um conjunto de classes que trabalham juntas, encorporando um padrão reutilizável para uma categoria de porblema. Ele dita a supra-estrutura da aplicação, descreve como responsabilidades são divididas entre vários componentes e como estes 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 encorporadas pelo framework não precisam ser reexaminadas e nem o código provido pelo *framework* precisa ser reescrito.

As metaheurísticas são em sua essência independentes de problemas. Elas são descritas como template que são completamente desenvolvidas quando o problema alvo se encontra formalmente definido. Além disso, as metaheurísticas são intrinsicamente dependentes de estratégias e parâmetros, os quais são completamente especificados após um processo de sintonia fina, de acordo com a instancia do problema a ser considerada. Os desevolvedores de aplicações muitas vezes 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 destes desenvolvedores. Portanto estes algoritmos se beneficiam fortemente de qualquer aspecto de engenharia de software que promova alta modularidade, reuso de software e interfaces de módulos bem definidas. Fatos estes que nos levam de volta a idéia de *framework*, que apresenta boa parte, senão todas as 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.

3 Metodologia

3.1 Formulação matemática do Problema de Roteamento de Veículos

Deixe $G = (V, A)$ ser um grafo direto onde $V = \{v_0, v_1,..., v_n\}$ 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 está um conjunto de veículos de diferentes capacidades, enquanto os vértices restantes correspondem às cidades ou consumidores. Cada consumidor v_i tem uma demanda não-negativa q_i e $q_0 = 0$. A cada arco (v_i, v_j) está associada uma distância não-negativa c_{ij} que representa a distância entre os consumidores (nós).

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

(a) Cada rota começa e termina no depósito;
(b) Cada cidade de $V\backslash\{v_0\}$ é visitada somente uma única vez por somente um veículo;
(c) A demanda total de cada rota não pode exceder a capacidade do caminhão;
(d) O tempo total gasto para percorrer a rota não pode ser maior que um limite LT dado como parâmetro;
(e) A distância total percorrida em uma rota não pode ser maior que um limite LD dado como parâmetro;

Formulação Matemática[10]

Variáveis principais:

t_i tempo de chegada no vértice i

wi tempo de espera no vértice i

$x_{ijk} \in \{0,1\}$ 1 se existe um arco dos vértices i para j utilizando um veículo k, 0 caso contrário $i \neq j; i, j \in \{0,1,2,...,N\}$

Parâmetros:

K número total de veículos

N número total de clientes

c_{ij} custo de viagem do vértice i para o vértice j

t_{ij} tempo de viagem entre os vértices i e j

m_i demanda do vértice i

q_k capacidade do veículo k

e_i tempo de chegada mais cedo no vértice i

l_i tempo de chegada mais tarde no vértice i

f_i tempo de serviço no vértice i

r_k tempo máximo permitido para a rota do veículo k

Minimizar: $\sum_{i=0}^{N} \sum_{j=0, j\neq i}^{N} \sum_{k=1}^{K} c_{ij} x_{ijk}$ (1)

sujeito a:

$$\sum_{k=1}^{K}\sum_{j=1}^{N} x_{ijk} \le K \qquad \text{para i=0} \qquad (2)$$

$$\sum_{j=1}^{N} x_{ijk} \quad = \quad \sum_{j=1}^{N} xjik \le 1 \qquad \text{para i = 0 e k} \in \{1,...,K\} \qquad (3)$$

$$\sum_{k=1}^{K}\sum_{j=0,j\neq i}^{N} x_{ijk} = 1 \qquad \text{para i} \in \{1,...,N\} \qquad (4)$$

$$\sum_{k=1}^{K}\sum_{i=0,i\neq j}^{N} x_{ijk} = 1 \qquad \text{para j} \in \{1,...,N\} \qquad (5)$$

$$\sum_{i=0}^{N} m_i \sum_{j=0.j\neq i}^{N} xi_{jk} \le q_k \qquad \text{para k} \in \{1,...,K\} \qquad (6)$$

$$\sum_{i=0}^{N}\sum_{j=0,j\neq i}^{N} x_{ijk}\left(t_{ij} + f_i + w_i\right) \le r_k \qquad \text{para k} \in \{1,...,K\} \qquad (7)$$

$$t_0 = w_0 = f_0 = 0 \qquad (8)$$

$$\sum_{k=1}^{K}\sum_{i=0,i\neq j}^{N} x_{ijk}\left(t_i + t_{ij} + f_i + w_i\right) \le t_j \qquad \text{para j} \in \{1,...,N\} \qquad (9)$$

$$e_i \le \left(t_i + w_i\right) \le I_i \qquad \text{para i} \in \{1,...,N\} \qquad (10)$$

A fórmula (1) é a função objetivo do problema. (2) especifica que existem no máximo K rotas saindo do depósito. A equação (3) certifica que todas as rotas começam e terminam no depósito central. As equações (4) e (5) definem que cada vértice de cliente pode ser visitado apenas uma vez por um veículo. A equação (6) é a restrição de capacidade. A equação (7) é a restrição do tempo máximo de viagem. As restrições de (8) a (10) definem a janela de tempo.

4 Modelagem

4.1 Histórico do desenvolvimento de um framework para otimização

O *framework* para otimização desenvolvido ao longo deste trabalho foi denominado JFFO. Um de seus objetivos é a construção de uma arquitetura que possibilite a reutilização de metaheurísticas. Porém, ao longo do trabalho constatou-se que mais estruturas poderiam ser reutilizadas como, por exemplo, estruturas de soluções, estruturas de movimentação.

Esta seção mostra como o framework proposto evoluiu no decorrer de seu desenvolvimento. Serão mostradas as estruturas de várias versões do JFFO, mostrando sua evolução e sua adequação aos problemas à medida que a complexidade destes aumentava.

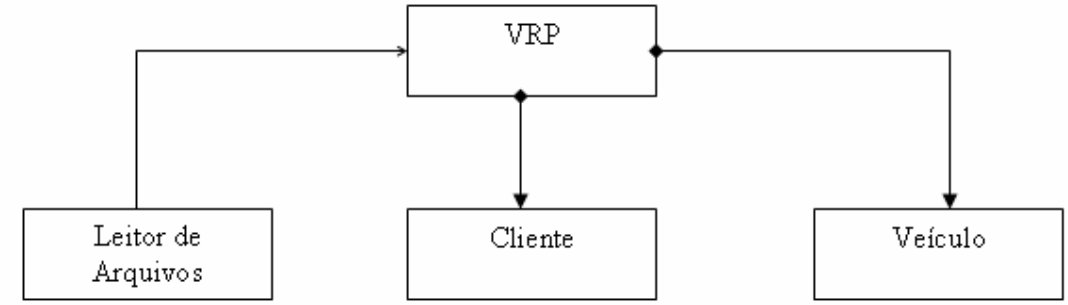

Figura 4.1. 1 Arquitetura de um aplicativo típico de otimização

A Figura 4.1.1 mostra o resultado de um trabalho preliminar no qual foi baseada a criação do JFFO. É um típico sistema para otimização na qual a classe chamada de `VRP` é responsável pela manutenção dos dados, processamento dos mesmos e também pelo processo de otimização.

A partir deste e de outros projetos, utilizando a experiência de programadores, foi-se generalizando características comuns até se chegar à primeira idéia de uma arquitetura para o JFFO.

A Figura 4.1.2 mostra a primeira arquitetura concebida com proposta de criação de um ambiente reutilizavel. Vale a pena mencionar que esta arquitetura não foi implementada, ela é apenas um modelo de referência. A primeira versão implementada para o JFFO será mostrada mais adiante.

As responsabilidades de cada classe desta versão são descritas a seguir.

`Repositório`: esta é a classe responsável por manter a base de dados, que são necessários para o processo de otimização, tais como matrizes, vetores e outras estruturas de dados que possam conter informações relevantes.

`Solução`: A classe solução é responsável por representar a solução corrente no processo de otimização.

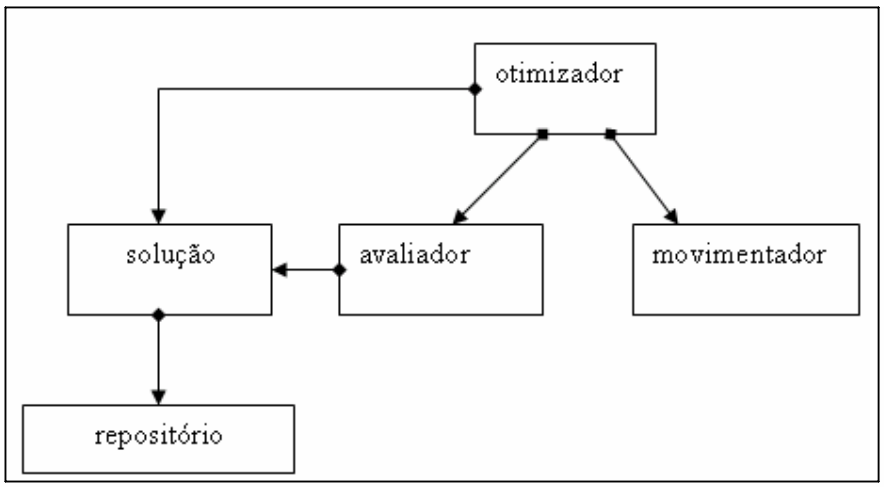

Figura 4.1. 2 JFFO modelo de referência

`Avaliador`: Esta classe é responsável por avaliar a qualidade das soluções durante o processo de otimização.

`Movimentador`: é a classe responsável por gerar "vizinhos" movimentando a solução atual.

`Otimizador`: é a classe responsável pelo processo de otimização. Nela está contido o algoritmo de otimização que o usuário escolheria pra a resolução do problema por ele trabalhado. Esta classe também é responsável por fazer o papel de mediador entre as classes solução, avaliador e movimentador.

Depois de avaliada a estrutura e constatadas algumas complicações desnecessárias neste modelo como, por exemlo, a necessidade da classe `solução` conhecer o repositório, uma vez que essa não o utiliza em momento algum, foi proposto um modelo alternativo.

A Figura 4.1.3 mostra a primeira versão implementada para o JFFO, que já continha dois algoritmos para otimização, um *Simulated Annealing* para problemas de minimização e outro para problemas de maximização. Também é mostrado como é feita a ligação entre as classes do *framework* e as classes de domínio.

Deve-se notar que a arquitetura do *framework* foi modificada, e com ela as responsabilidades de algumas classes. A única classe que não teve sua responsabilidade alterada foi a classe de armazenamento de dados que agora se chama `Store`.

A figura também mostra a ligação das classes do *framework* com as classes de domínio específico para o PRV.

As responsabilidades de cada classe serão descritas a seguir.

`Store`: esta é a classe responsável por manter a base de dados, que são necessários para o processo de otimização, tais como matrizes, vetores e outras estruturas de dados que possam conter informações relevantes.

`Solution`: A classe solução é responsável por representar a solução corrente no processo de otimização. Também é responsável por fazer a intermediação entre a classe Optimizer e as classes Avaliador e Movimentador.

`Avaliador`: Esta classe é responsável por avaliar a qualidade das soluções durante o processo de otimização.

`Movimentador`: é a classe responsável por gerar "vizinhos", movimentando a solução atual.

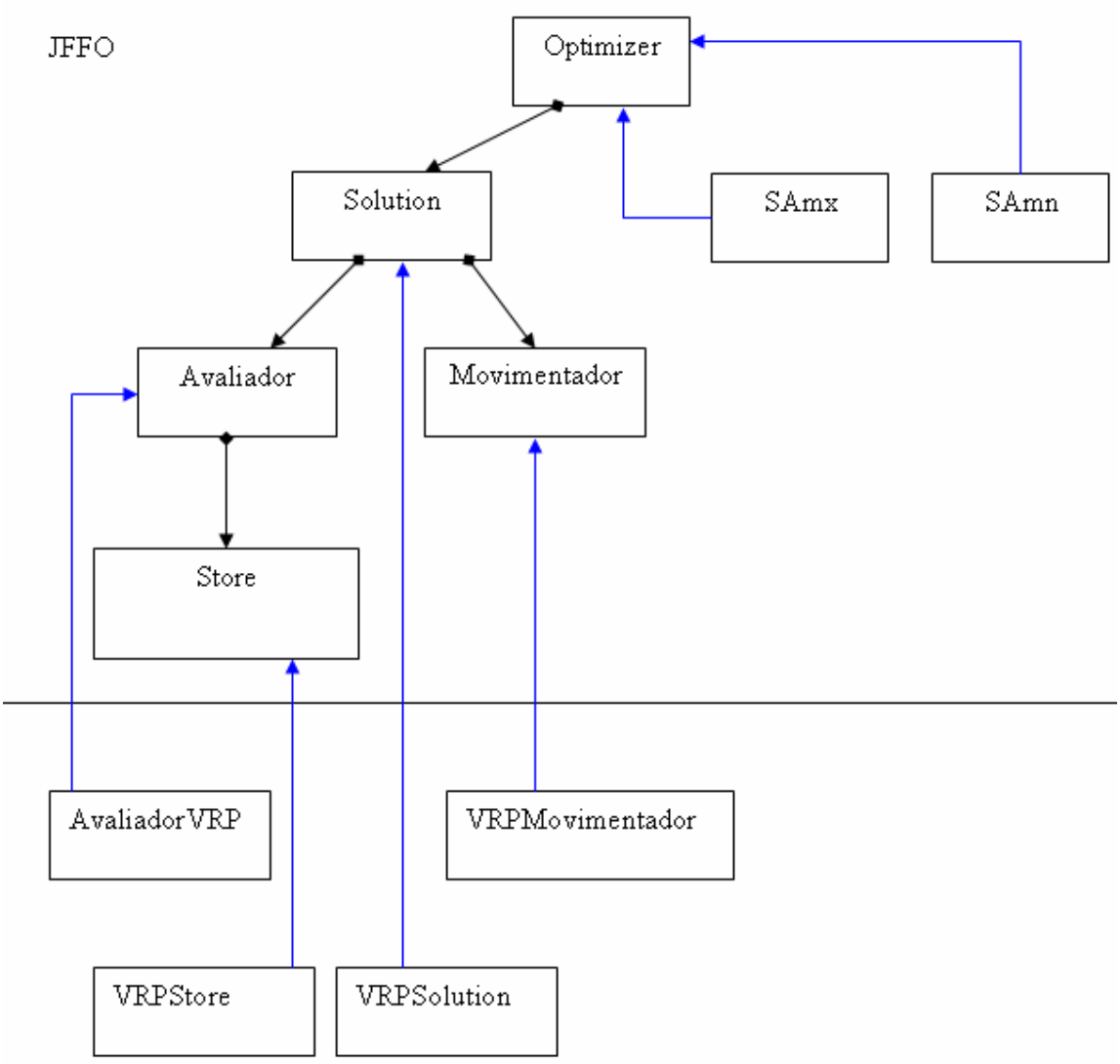

Figura 4.1. 3 JFFO primeira versão

`Optimizer`: é a classe responsável pelo processo de otimização. Nela está contido o algoritmo de otimização que o usuário escolheria pra a resolução do problema por ele trabalhado.

Este modelo mostrou-se adequado para o uso do algoritmo *Simulated Annealing,* porém, mostrou-se pouco adequado à inclusão de novas metaheurísticas, além disso, mostrou-se demasiadamente complexo para o uso simultâneo de várias estruturas de vizinhaça.

Para corrigir os problemas mencionados anteriormente foi proposta uma segunda arquitetura, apresentada pela Figura 4.1.4 .

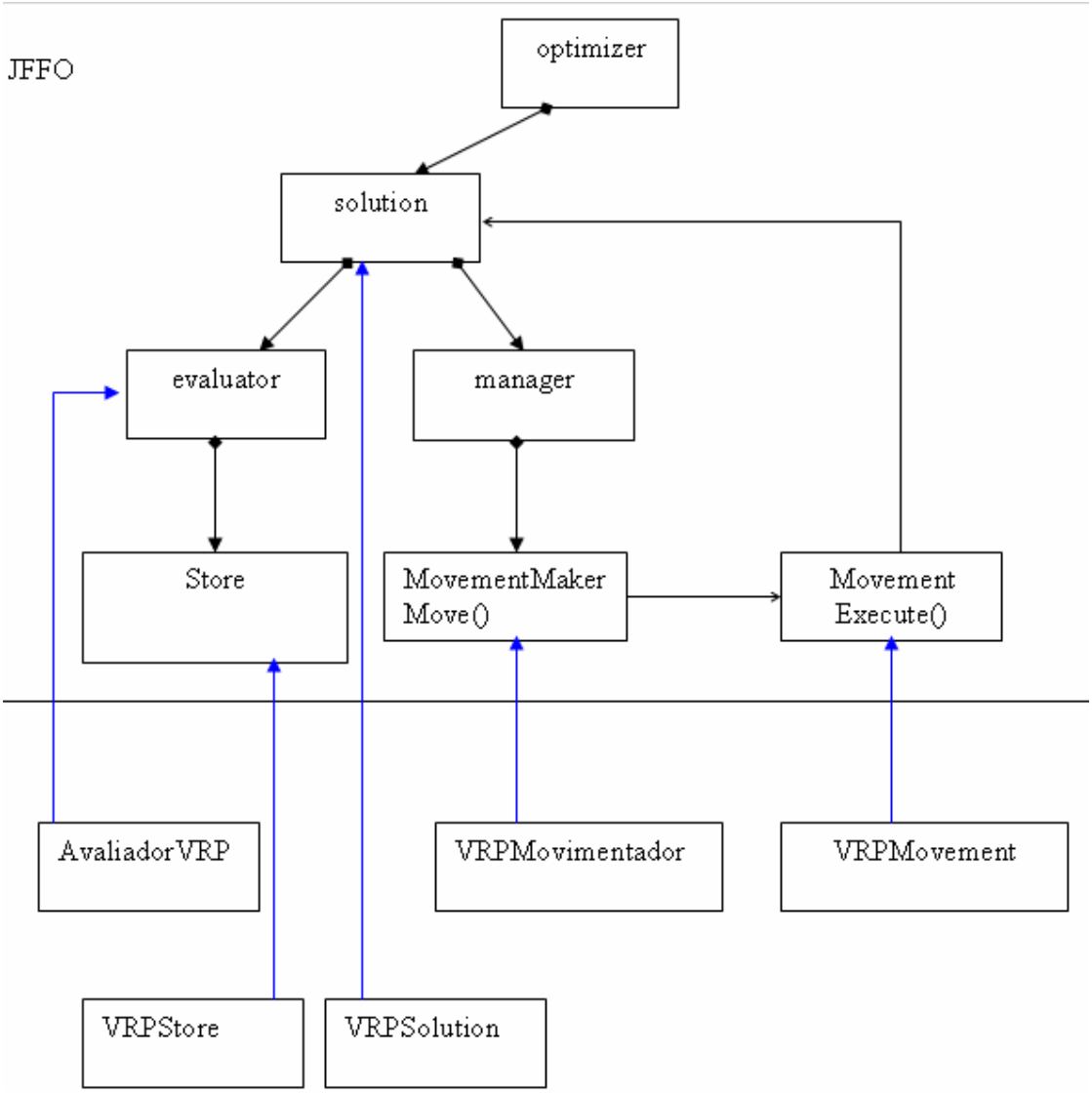

Figura 4.1. 4 JFFO segunda versão

A presença da classe `Manager` permitiu o uso simultâneo de várias estruturas de vizinhança e a presença da classe `Movement` permitiu a inclusão de pelo menos uma metaheurística, a *busca tabu*. A inclusão de outras metaheurísticas está sendo preparada e será objeto de estudos futuros.

A seguir as responsabilidades de cada classe.

`Manager`: é responsável por gerenciar e manter simultaneamente uma ou mais estruturas de vizinhaça.

`MovementMaker`: responsável por gerar vizinhos, movimentando e desfazendo movimentos, e por construir objetos do tipo Movement.

`Movement`: é responsável por representar um movimento que foi ou será executado. Esta classe também tem como responsabilidade saber se reproduzir e se desfazer, ou seja, ela também é responsável por gerar vizinhos através de movimentações.

As demais classes permanecem com a mesma responsabilidade que tinham na versão anterior. A única diferença é a nomenclatura que foi adequada ao inglês, para fins de uniformização.

O modelo apresentado possui uma redundância de responsabilidades entre as classes `MovementMaker` e `Movement`. Tal redundância foi gerada por adaptação da versão anterior para esta versão. Nota-se claramente que a presença desta redundância aumenta a complexidade de entendimento da estrutura do JFFO e também entra em discordância com regras de engenharia de software e da boa prática de programação.

Este modelo também se mostra insuficiente para metaheurísticas que trabalham com uma população (pool) de soluções, uma vez que até agora, as metaheurísticas trabalhadas possuíam apenas uma única solução interna.

Para resolver este problema, um terceiro modelo foi criado.

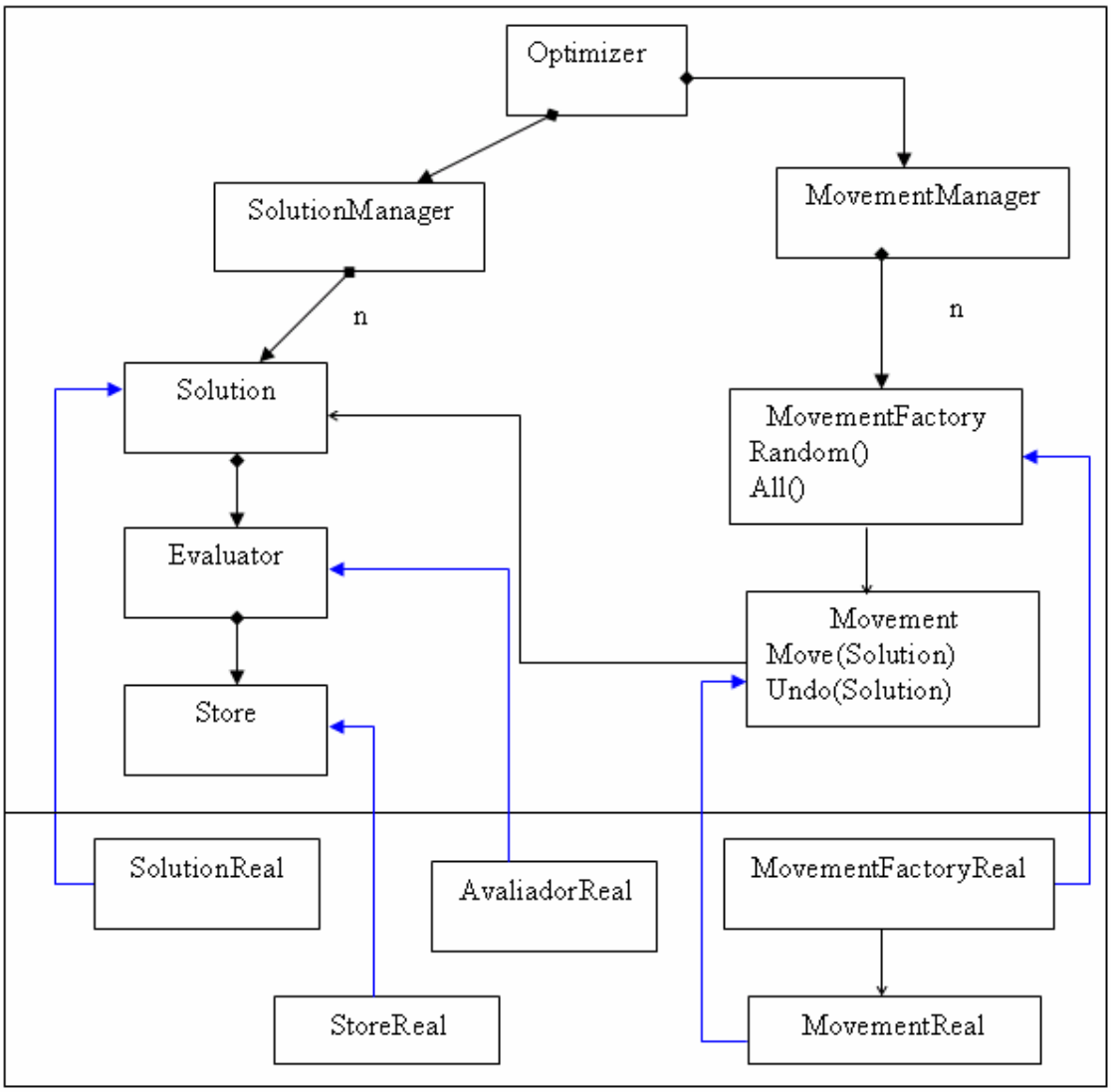

Figura 4.1. 5 JFFO terceira versão

O modelo apresentado pela Figura 4.1.5 é o modelo atual do JFFO. Este modelo corrige os problemas da versão anterior e apresenta uma estrutura promissora para futuras inclusões de novas metaheurísticas, estruturas de movimentação e novos modelos de solução.

As responsabilidades de cada classe serão 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 pool 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`: Resonsável por construir os objetos para um determinado tipo de solução. Dentro da estrutura do framework ela trabalha como fabrica abstrata, sendo responsabilidade de suas subclasses construir os movimentos adequados. A presença desta classe é importante para possibilitar o uso de várias estruturas de vizinhaça. Assim, cada `MovementFactory` representa uma estrutura de vizinhaça.

`Optimizer`: Responsável pelo processo de otimização. Nesta versão, esta classe também faz o papel de mediador entre as soluções e as classes movimentadoras.

`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 diverssas estruturas de vizinhaça.

4.2 JFFO – *Java Framework For Optimization*

O JFFO está dividido em cinco pacotes. Os pacotes, suas responsabilidades e as classes que possuem até o momento, serão mostrados a seguir.

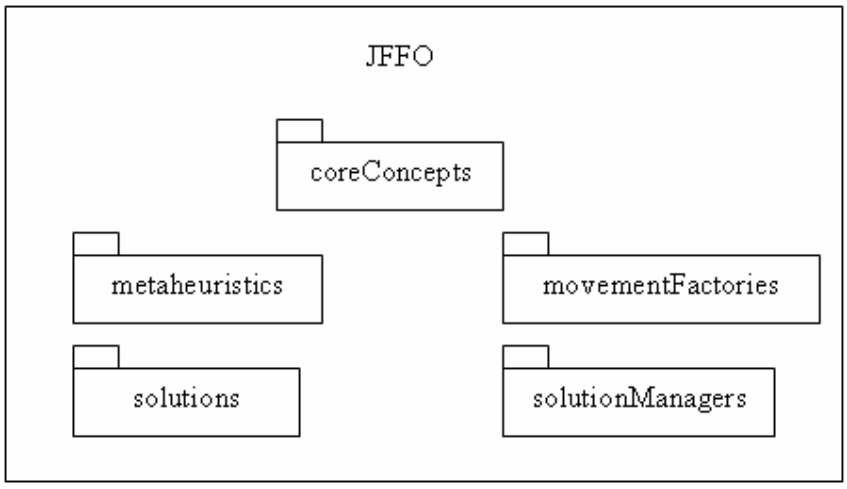

Figura 4.2. 1 Pacotes do JFFO

4.2.1 O pacote `coreConcepts`.

Este pacote contém os conceitos principais do *framework*. Todo o *framework* depende, de alguma forama, das classes (pelo menos uma) contidas neste pacote.

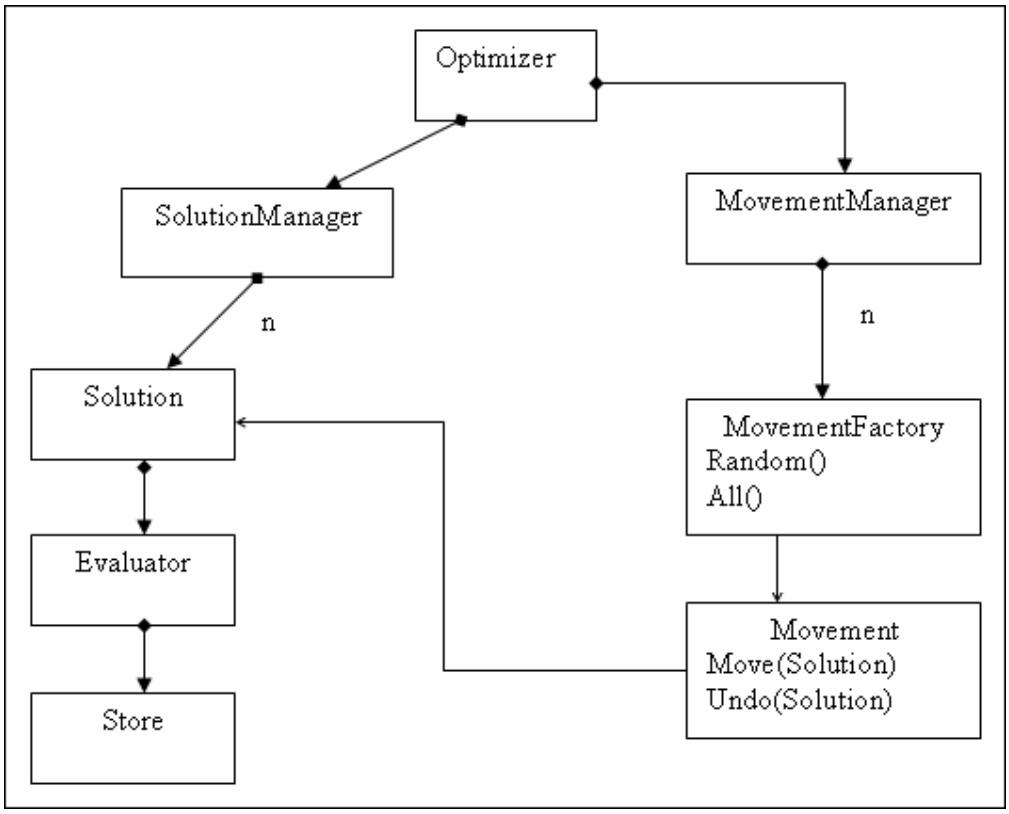

Figura 4.2.1. 1 Pacote coreConcepts

4.2.2 O pacote `metaheuristics.`

Este pacote contém as classes que são otimizadores reais, ou seja, classes reais que extendem a classe `Optimizer`, que é abstrata. No momento, existem apenas duas metaheurísticas implementadas. *Tabu search* e *Simulated Annealing*, cada uma com suas variantes para problemas de minimização e maximização.

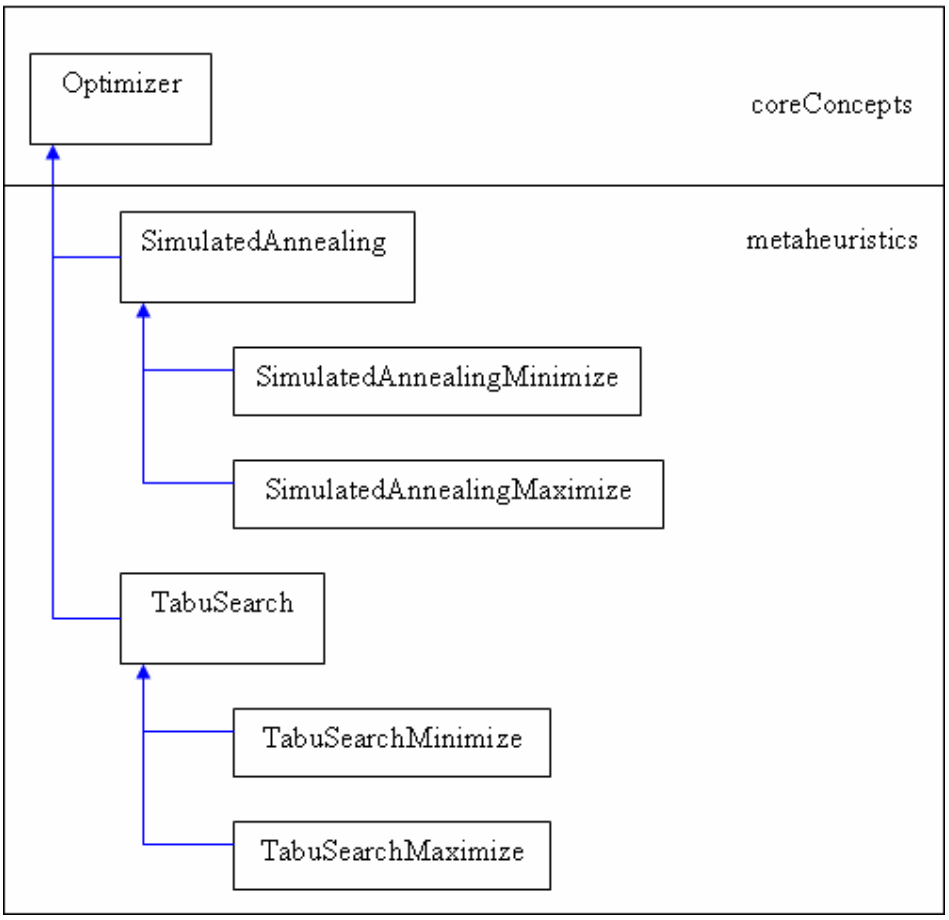

Figura 4.2.2. 1 Pacote metaHeuristics

4.2.3 O pacote `solutions`.

Este pacote possui as possíveis estruturas de solução que podem ser utilizadas para o desenvolvimento de aplicações. No momento, apenas uma estrutura é oferecida. A presença de estruturas de solução pré-construida não é, ao contrário do que parece, um limitador para o usuário do JFFO. As estruturas contidas neste pacote são sim uma forma de fornecer uma forma conhecida e já testada de representar uma solução. O usuário tem toda a liberdade de construir suas próprias estruturas de solução. Contudo, para que as estruturas criadas pelo usuário possam interagir com o JFFO elas precisam respeitar a interface da classe `Solution`, presente no pacote `coreConcepts`.

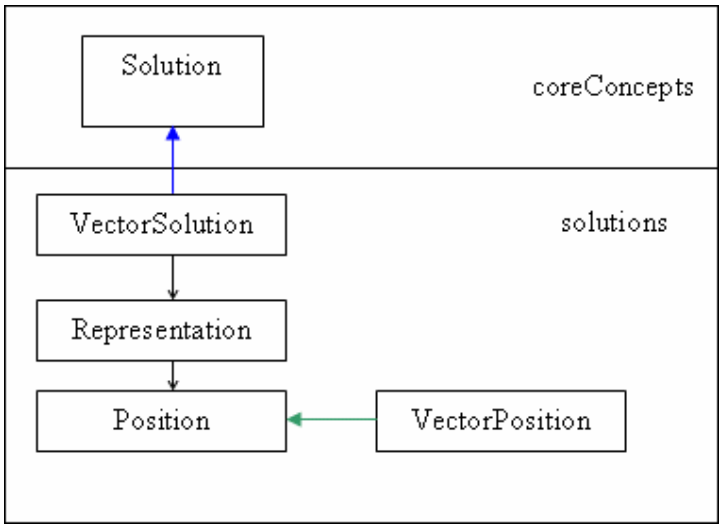

Figura 4.2.3. 1 Pacote solutions

A estrutura fornecida atualmente é a de solução por vetor de alocação dinâmica. Esta esturura é implementada pela classe `VectorSolution`. Esta estrutura usa como auxiliaras a classe `Representation`, que responsável por dizer o que uma posição dentro do vetor representa, e onde o objeto representado se encontra dentro da base de dados do problema. Para armazenar a informação de onde se encontra o objeto representado, a classe `Representation` utiliza a classe `Position`. Apenas uma estrutura para armazenamento de localidade está implementada atualmente, que é a `VectorPosition`, esta classe serve para guardar informações sobre a posição de objetos que se encontram armazenados em outros vetores dentro da base de dados.

4.2.4 O pacote `movementFactories`.

Este pacote possui estruturas de movimentação, como fabricas de movimentos e movimentos para as estruturas de solução providas pelo JFFO. No momento, duas estururas de movimentação estão disponíveis para o modelo de solução implementado por vetor dinâmico, representado no pacote `solutions` pela classe `VectorSolution`. Nesta seção, estas estruturas serão apenas apresentadas. Uma descrição mais completa de seu funcionamento será apresentada mais adiante.

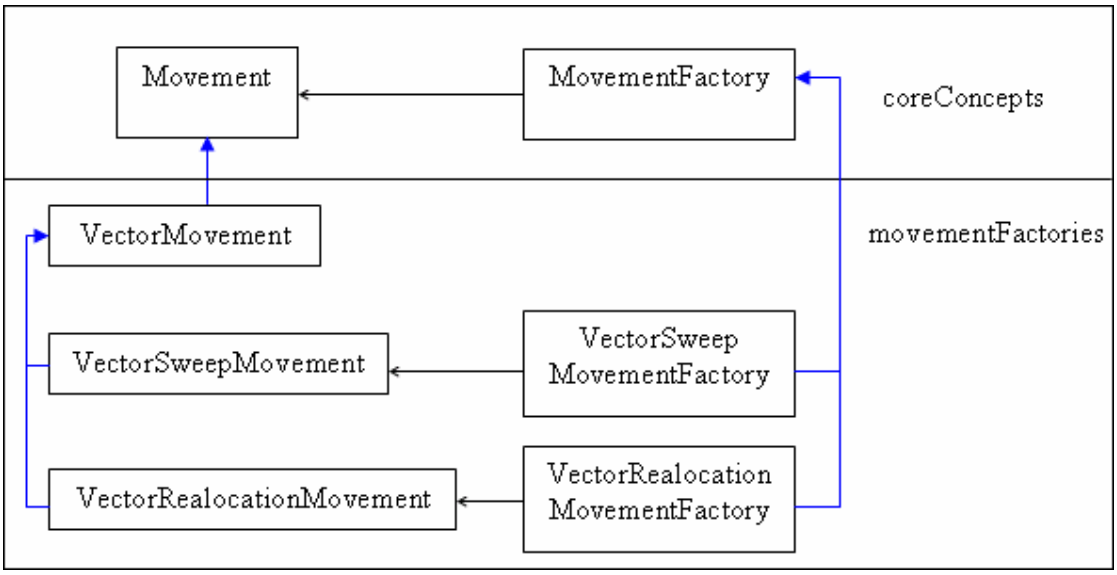

Figura4.2.4. 1 Pacote movementFactories

A seguir, será apresentada a descrição de cada classe.

`VectorMovement`: Classe responsável por armazenar informações para a execução de movimentos em uma estrutura de vetor como, por exemplo, qual posição será movida.

`VectorSweepMovement`: Responsável por fazer o movimento de troca entre posições do vetor. Uma descrição mais detalhada deste tipo de movimento será apresentada mais adiante.

`VectorSweepMovementFactory`: Responsável por construir objetos do tipo `VectorSweepMovement` a partir de uma solução. Para ilustrar a responsabilidade desta classe, pode-se dar um exemplo: Uma solução representada por vetor que tenha 30 posições não pode ter nenhuma movimentação para a posição de numero 54. É responsabilidade desta classe construir apenas movimentos que possam ser executados.

`VectorRealocationMovement`: Responsável por fazer o movimento de realocação de uma posição do vetor. Uma descrição mais detalhada deste tipo de movimento será apresentada mais adiante.

`VectorRealocationMovementFactory`: Responsável por construir objetos do tipo `VectorRealocationMovement` a partir de uma solução. Para ilustrar a responsabilidade desta classe, o mesmo exemplo dado para a classe `VectorSweepMovementFactory` pode ser aplicado.

4.2.5 O pacote `solutionManagers`.

Este pacote é responsável por conter os gerenciadores de solução reais, uma vez que o apresentado no pacote coreConcepts é abstrato. Como nos modelos para soluções, os gerenciadores apresentados aqui não são uma limitação para o usuário do *framework*, mas sim uma forma de prover gerenciadores já testados para as situações mais comuns. O usuário pode, como nas soluções, produzir seus próprios gerenciadores para interagir com o restante do *framework*. No momento, apenas um gerenciador está implementado. O estudo para a inclusão de novos gerenciadores será objeto de trabalhos futuros.

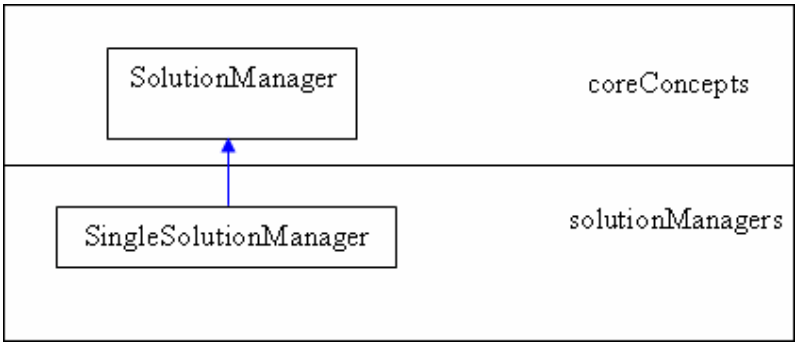

Figura 4.2.5. 1 Pacote solutionManagers

A classe `SingleSolutionManager` é a classe responsável por gerenciar uma única solução. Este gerenciador pode ser utilizado por metaheurísticas que trabalham internamente com uma única solução como, por exemplo, *Simulated Annealing* e *Busca Tabu*.

4.3 Estrutura

O Projeto está dividido em uma série de classes as quais serão mostradas a seguir juntamente com suas funcionalidades. Primeiro será mostrada como foi feita a especialização do *framework* para o PRVFHJT.

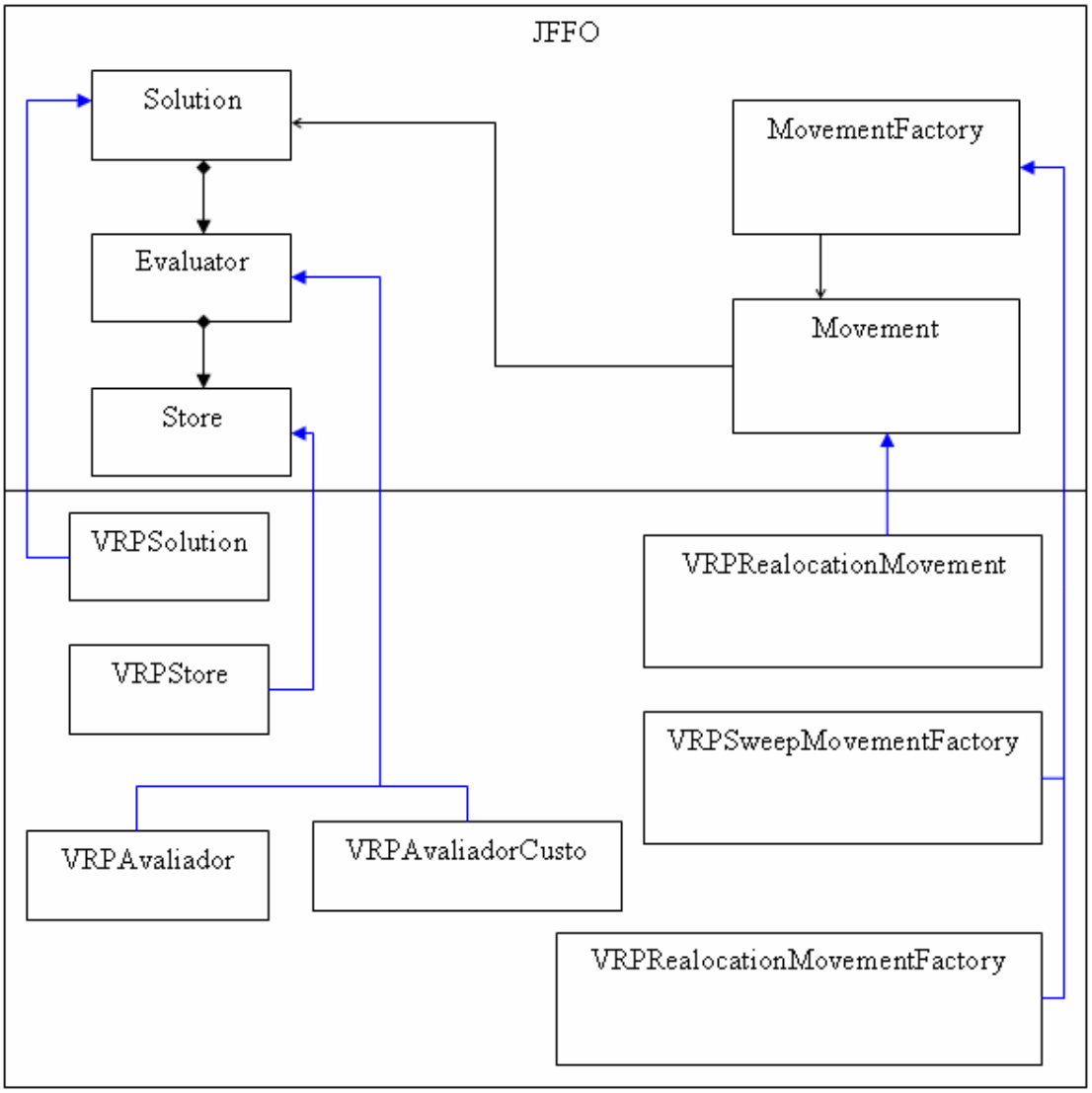

Figura 4.3. 1 - Ligação entre o framework e as classes de domínio

A Figura 4.3.1 mostra como é feita a ligação indireta entre as classes de domínio e o pacote `coreConcepts`. As classes do *framework* que foram especializadas são mostradas na parte superior da figura. As classes de domínio específico são as da parte inferior da figura.

Esta ligação se dá de forma indireta para as classes `VRPRealocationMovement`, `VRPRealocationMovementFactory`, `VRPSweepMovementFactory` e

VRPSolution porque na realidade, as classes de domínio estendem classes dos demais pacotes. A ligação direta entre o *framework* e estas classes de domínio do problema será mostrada mais adiante. As ligações entre as classes Store e VRPStore e a ligação entre as classes VRPAvaliador, VRPAvaliadorCusto e Evaluator, também mostradas na figura, são ligações diretas, uma vez que o *framework* ainda não provê classes para especializar estas ligações.

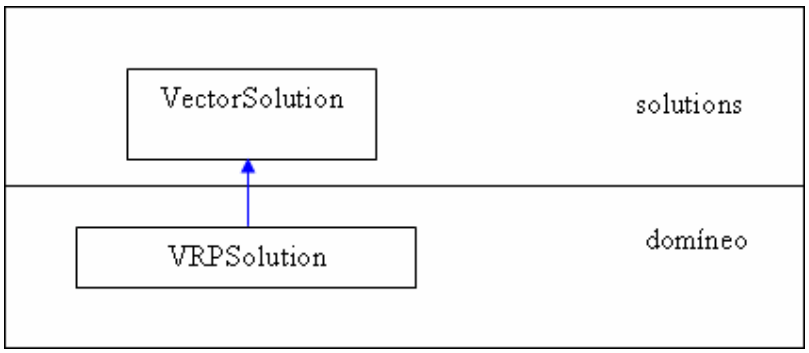

Figura 4.3. 2 - Ligação da Solução de domínio com a do framework

A classe VRPSolution é a especialização da classe VectorSolution para o PRVFHJT.

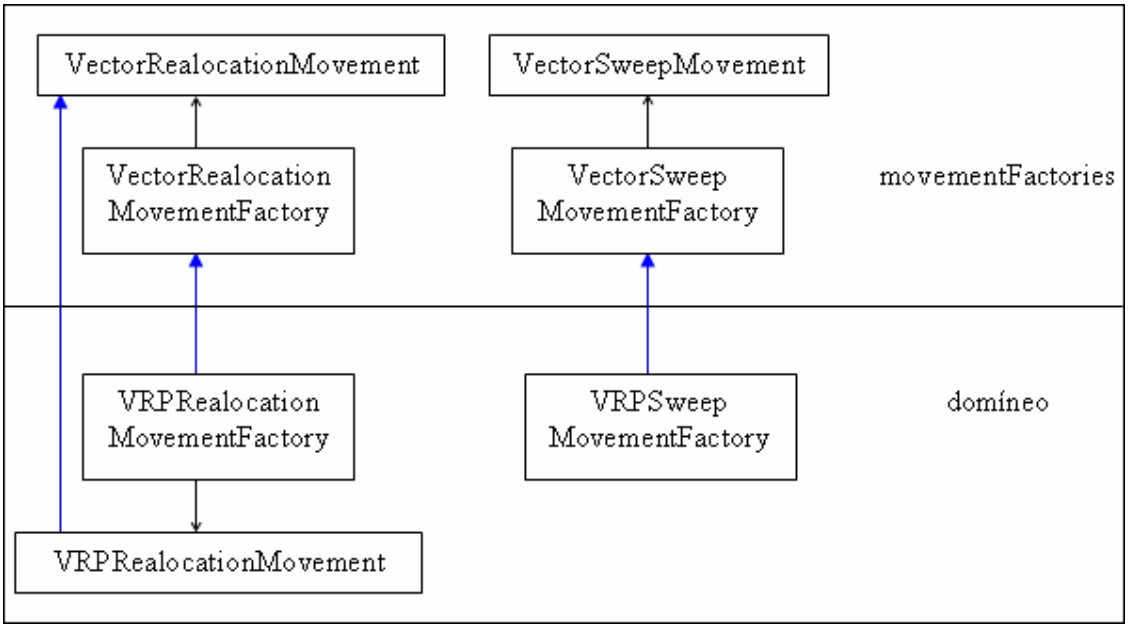

Figura 4.3. 3 -Ligação das classes de movimentação

As classes VRPRealocationMovementFactory, VRPSweepMovementFactory e VRPRealocationMovement são as especializações das classes VectorRealocationMovementFactory, VectorSweepMovementFactory e VectorRealocationMovment respectivamente. A classe VectorSweepMovement foi utilizada sem a necessidade de especialização.

A seguir, serão apresentadas todas as classes da aplicação construída para a validação do JFFO e suas responsabilidades.

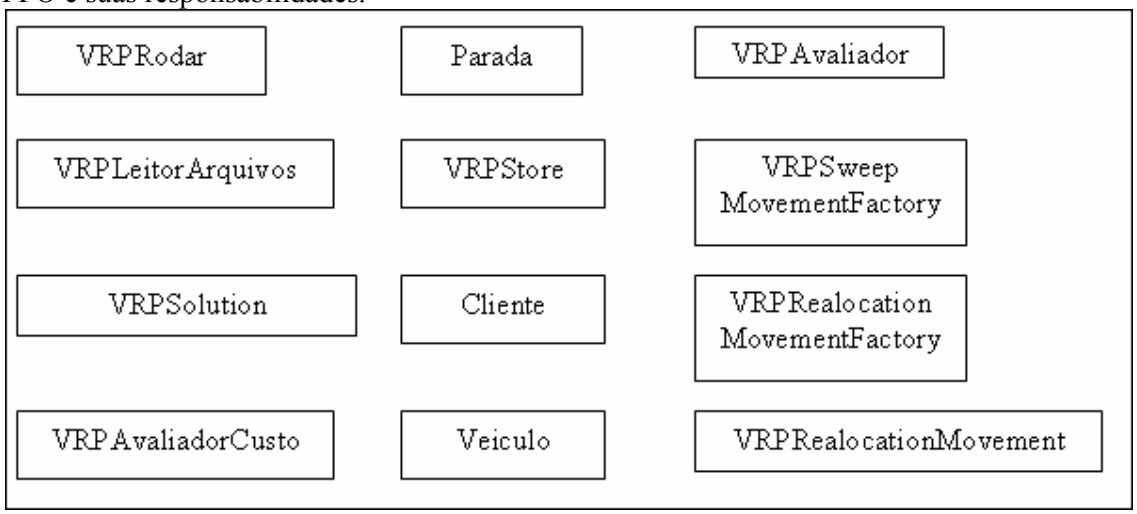

Figura 4.3. 4 Classes da aplicação

A classe `LeitorArquivos` é responsável por ler os dados que estão nos arquivos, transformá-los para o formato desejado e enviá-los para a classe `VRPStore`, onde ficarão armazenados para uso futuro.

A classe `Parada` é responsável por armazenar os dados referentes às paradas feitas pelos motoristas além dos atendimentos (Almoço e jantar, por exemplo).

A classe `Cliente` é responsável por armazenar os dados de um cliente como nome, demanda, janelas de tempo, etc.

A classe `Veículo` é responsável por guardar as informações de um veículo como rótulo, capacidade, quantidade de carga atual, etc.

A classe `VRPStore` é responsável por armazenar dados importantes para o processo de otimização como por exemplo, a matriz que contém as distâncias entre cada par de clientes.

A classe `VRPSolution` é responsável por representar soluções para o PRVFHJT. Cada instancia desta classe representa uma possível solução para o problema.

A classe `VRPAvaliador` é responsável por avaliar as soluções durante o processo de otimização.

A classe `VRPAvaliadorCusto` também é responsável por avaliar as soluções. Porém ela não é utilizada durante o processo de otimização. O que ela faz é avaliar uma solução de um ponto de vista unicamente financeiro. A informação financeira também é utilizada dentro do processo de otimização pela classe `VRPAvaliador`. Esta classe é utilizada para ilustrar de uma maneira mais intuitiva o processo de otimização.

As classes `VRPSweepMovementFactory` e `VRPRealocationMovementFactory` são responsáveis por construir movimentos dos tipos `VectorSweepMovement` e `VRPRealocationMovement` respectivamente. Estas fábricas tiveram que ser especializadas pelo fato de o modelo de solução adotado possui restrições para a movimentação de certas posições. Logo, alguns movimentos não podem ser construídos.

A classe `VRPRealocationMovement` é responsável por realizar movimentos de realocação durante o processo de otimização. Ela teve que ser especilizada pelo fato de que o método de comparação da superclasse se mostrou ineficente para a resolução do PRVFHJT.

Por último, a classe `VRPRodar`. Esta classe é responsável por construir os objetos necessários, ligá-los de maneira conveniente e chamar o processo de otimização que será realizado por objetos do tipo `SimulatedAnnealingMinimize` e `TabuSearchMinimize`, que também são construídos pela classe `VRPRodar`.

4.4 Base de dados

A base de dados, que é implementada pela classe `VRPStore`, contém os dados necessários para a resolução do problema. Estes dados são divididos em uma série matrizes, vetores, e valores que serão descritos adiante.

Os dados relativos aos clientes encontram-se em um vetor de clientes, o qual pode ser referenciado para consulta a qualquer momento, no entanto não sendo modificado durante a resolução do problema. De tal vetor podem ser extraídas informações tais como numero de clientes, as demandas, tempo de atendimento e as janelas de tempo de cada cliente.

Os dados relativos aos veículos se encontram em um vetor de veículos, o qual pode ser referenciado para consulta a qualquer momento da resolução do problema e para modificação durante um período na resolução do problema. Deste vetor podem ser extraídas informações tais como a quantidade de veículos, a capacidade, o custo operacional, o custo quilometrado, e quantidade de mercadoria no veículo em um dado instante para cada um dos veículos.

Os dados relativos às paradas que o motorista pode fazer durante o percurso da rota são armazenadas em um vetor de paradas, o qual contém informações tais como, numero de paradas, tempo de início e duração de cada uma das paradas.

Há duas matrizes na modelagem. Uma que contém todas as distâncias entre os pares de clientes, e uma matriz com o tempo de viagem entre os pares de clientes. Ambas as matrizes são preenchidas durante um pré-processamento onde o método dos quadrados mínimos é utilizado para calcular a distância entre os clientes, a partir destas distâncias e da velocidade média, que é um parâmetro de entrada, é preenchida a matriz de tempos.

Alguns dados adicionais são necessários para a resolução do problema, estes dados são os limites de tempo e distância de cada rota, e o limite de tempo das horas extras. Tais dados são armazenados em variáveis.

4.5 Modelagem da solução

A modelagem utilizada para a resolução do problema mostra-se adequada, do ponto de vista da generalidade e da expansividade, pois, trata de maneira conveniente o problema e permite que no futuro, novos elementos sejam incorporados à mesma. Este atributo, classificado em Cordeau et al. (2002) [9] como flexibilidade, é um fator positivo para uma modelagem.

A solução, implementada pela classe `VRPSolution`, internamente é representada por um vetor de objetos de um tipo simples chamado `Representation`, que serve para criar uma indireção entre a modelagem da solução e a modelagem do restante dos dados.

Um objeto do tipo `Representation` contém, neste caso especifico, dois dados; um significa o tipo de instância que este objeto representa. No nosso caso 0 (zero) para veículos e 1 (um) para clientes.

Como a solução é um vetor, assim ficaria um exemplo com um veículo e 3 clientes

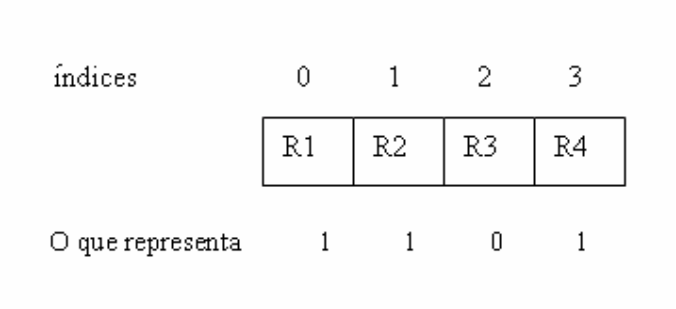

Figura 4.3. 5 Objetos e o que representam

Um objeto do tipo representação também possui um índice que aponta para qual objeto ele representa. Este índice indica onde na base de dados se encontra o objeto o qual este objeto representa.

Figura 4.3. 6 Objetos, oque representam e onde

O esquema acima mostra que o objeto R1 representa um cliente, e que este cliente se encontra na posição 2 da base de dados. Por sua vez, o objeto R3 representa um veículo que se encontra na posição zero na base de dados.

A posição de cada representação também tem significado. Se um cliente será atendido pelo veículo mais próximo com índice menor que o dele. Em outras palavras, um veículo atenderá todos os clientes que se interpõem entre ele e o próximo veículo na solução. No exemplo anterior, seguindo o que foi dito, o cliente representado por R4 seria atendido pelo veículo representado por R3. Segue outro exemplo.

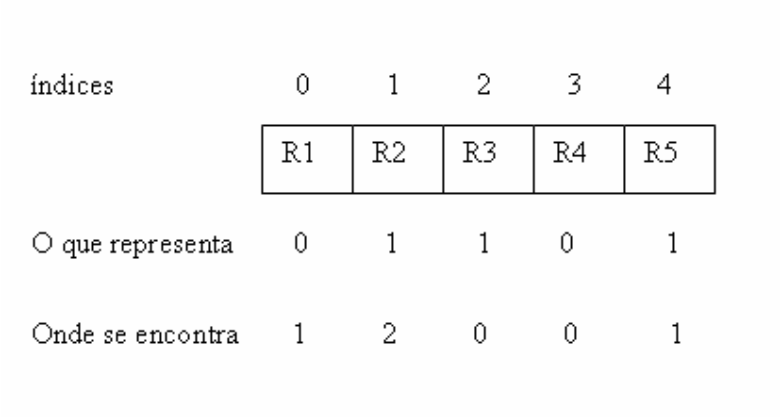

Figura 4.3. 7 Um Exemplo de solução

No exemplo acima, os clientes representados por R2 e R3 são atendidos pelo veículo representado por R1 enquanto, o cliente representado por R5 é atendido pelo veículo representado por R4.

4.6 Estrutura de vizinhaça

São duas as estruturas de vizinhança adotadas para o modelo, uma estrutura baseada em troca e outra baseada em realocação.

A estrutura de troca para o modelo adotado é bem simples, qualquer permutação de duas posições na solução é considerada um vizinho. Exemplo:

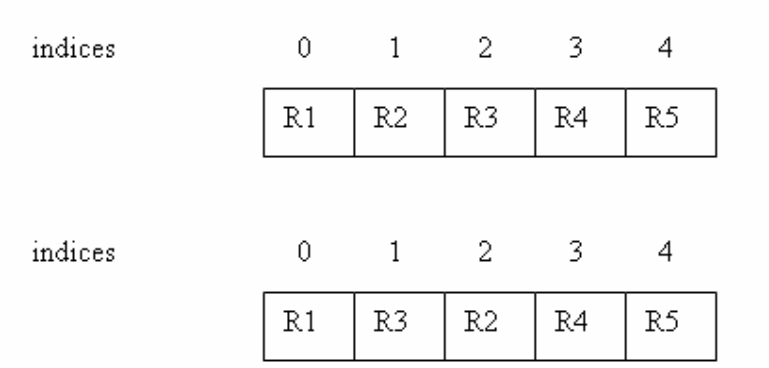

Figura 4.4 1 Soluções vizinhas

Note que a diferença entre as duas soluções é a troca de posições entre R2 e R3.

Assim modelada, a vizinhança abrange implicitamente 4 tipos diferentes de variação da solução:

Intra-Rota – quando dois clientes da mesma rota são permutados, mudando a ordem de visita dos mesmos.

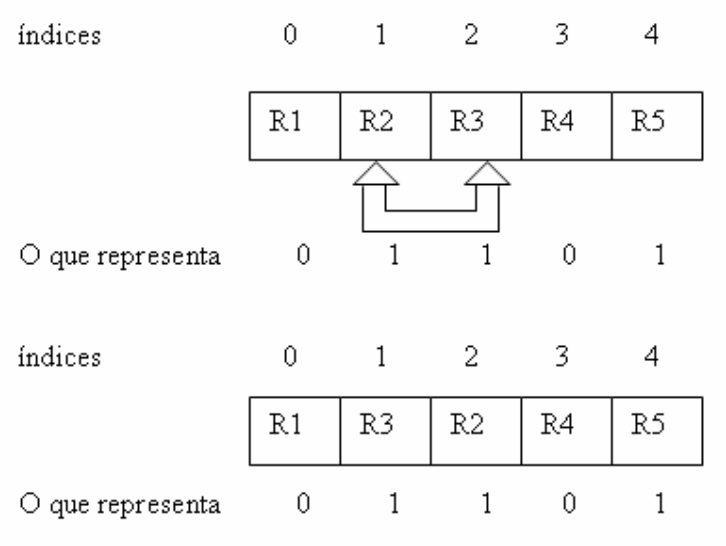

Figura 4.4 2 Movimentação Intra-Rota

Inter-Rota – quando dois clientes de duas rotas diferentes são permutados, mudando assim, duas rotas.

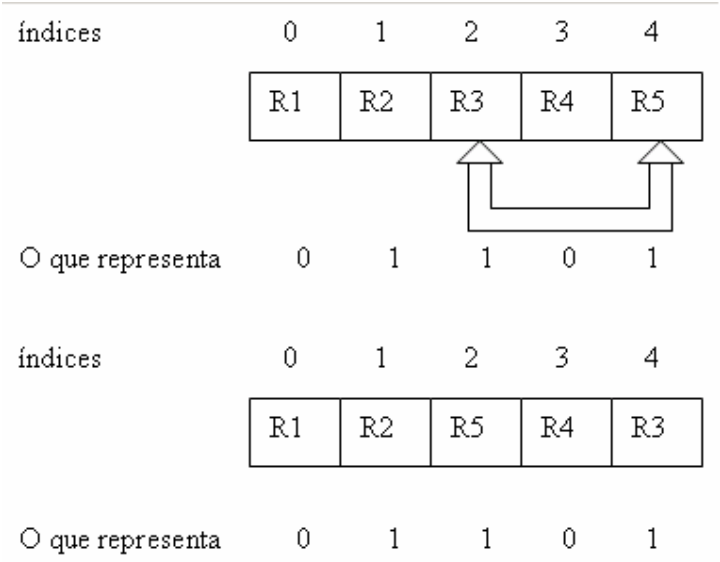

Figura 4.4 3 Movimentação Inter-Rota

Veículo/Cliente – quando um cliente e um veículo são permutados, fazendo com que uma rota diminua e uma outra aumente.

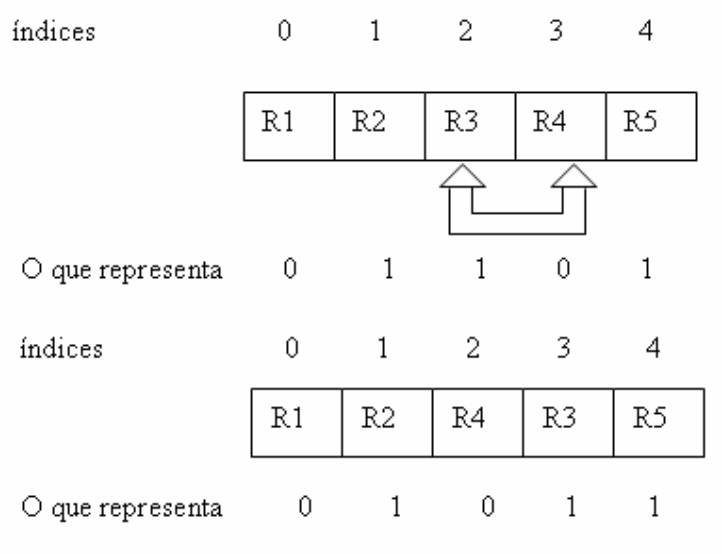

Figura 4.4 4 Movimentação Veículo/Cliente

Troca de rota – quando dois veículos são permutados, fazendo que suas rotas anteriores sejam percorridas por outro veículo, possivelmente de capacidade e custos diferentes.

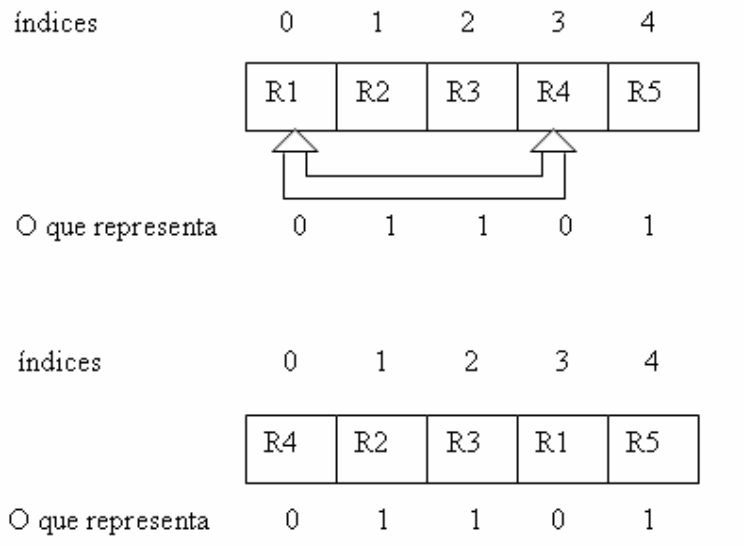

Figura 4.4 5 Movimentação Troca de Rota

Movimentos desta estrutura de vizinhaça são executados pela classe `VectorSweepMovement`. Para o PRVFHJT usou-se uma fábrica especifica para produzir estes movimentos. Esta fábrica é implementada pela classe `VRPSweepMovementFactory` e foi construída pelo fato de que a posição de índice zero da solução deve sempre ser ocupada por um veículo. Logo, movimentos que troquem o veículo de posição zero com um cliente qualquer não devem ser produzidos. O veículo de posição zero só pode ser trocado por outro veículo.

A estrutura de realocação também é simples, dadas duas posições, uma chamada de "posição anterior" e outra de "nova posição", desloca-se o elemento da "posição anterior" para a "nova posição", mantendo a ordem do restante dos elementos e fazendo os deslocamentos nessessários.

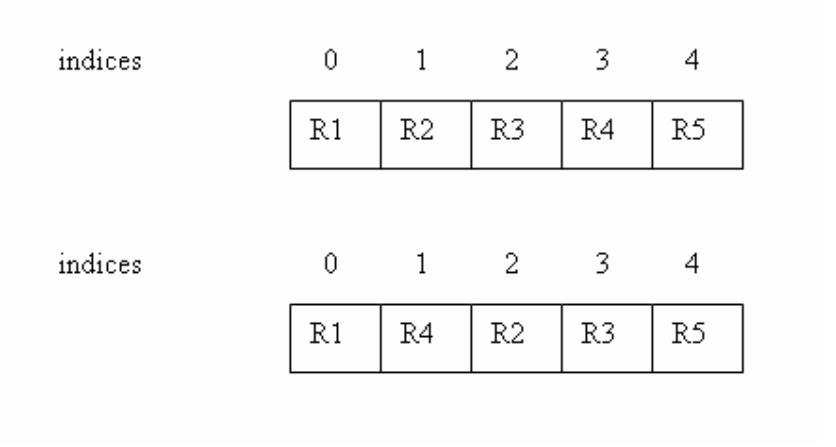

Figura 4.4 6 Soluções vizinhas

Na figura acima a posição de índice 3 é a "posição anterior" e a posição de índice 1 é a "nova posição". O movimento consiste em incerir o elemento da posição 3 na posição 1, mantendo a mesma ordem no restante da solução, é necessário deslocar as demais posições (1 e 2) para direita.

Assim modelada a vizinhança modela 3 tipos de variantes da solução.

Intra-Rota - quando as posições escolhidas pertencem à mesma rota, isso faz com que se modifique a ordem de vizita dos clientes.

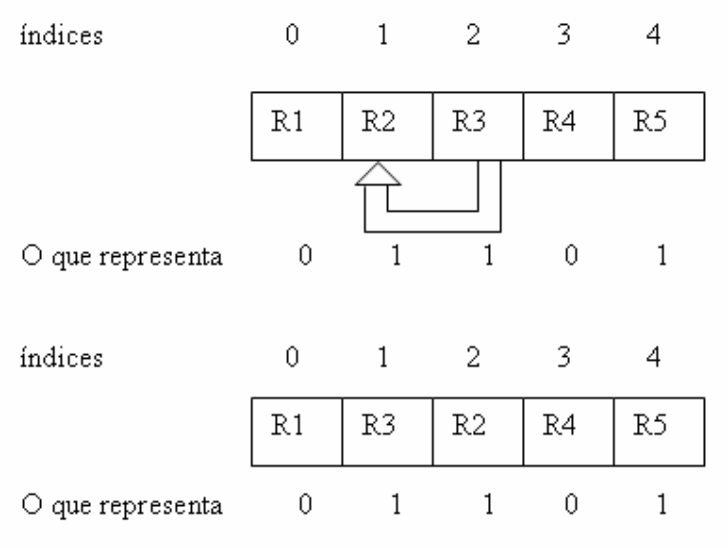

Figura 4.4 7 Movimento Intra-Rota

Inter-Rota - quando posições de rotas diferentes são escolhidas, isso faz que um cliente saia de uma rota e vá para outra.

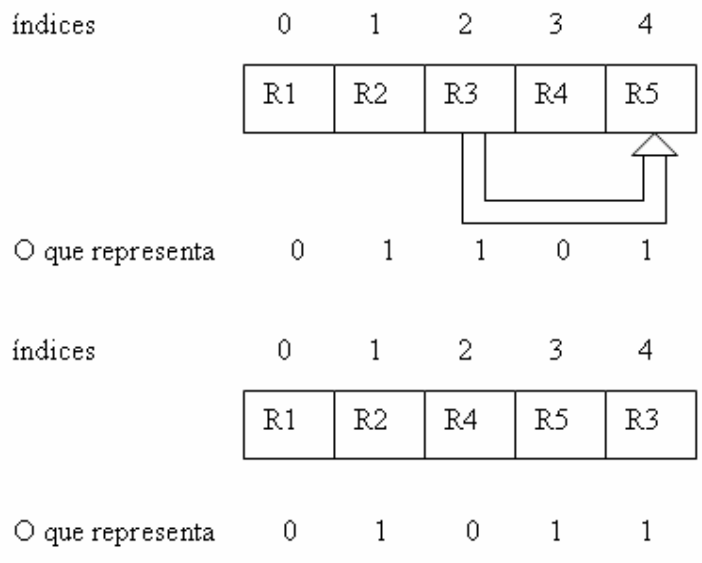

Figura 4.4 8 Movimento Inter-Rota

Note que, desta vez, os elementos restantes não foram deslocados para a direita, mas sim para a esquerda. Isto ocorre sempre que a "posição anterior" tem índice menor que a "nova posição". Quando a "posição anterior" tem índice maior que a "nova posição", os demais elementos são deslocados para a direita.

Movimentação de veículo - ocorre quando o elemento da “posição anterior” escolhida é um veículo. Neste caso, independente da “nova posição” escolhida, ocorrerá uma mudança drástica, Com a possível inclusão de vários clientes na rota deste veículo e também com a possível retirada de vários clientes da rota de outro veículo.

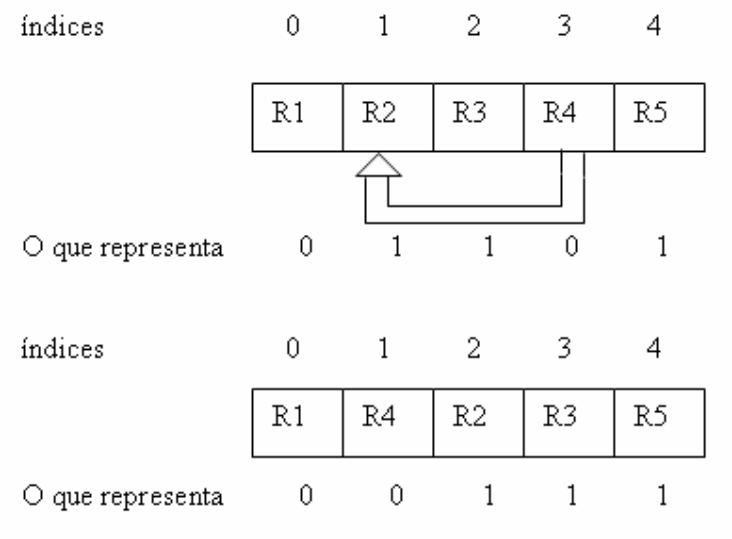

Figura 4.4 9 Movimento de veículo

Movimentos desta estrutura de vizinhaça são executados pela classe `VRPRealocationMovement`. Esta classe foi especializada a partir da classe `VectorRealocationMovement`. A idéia da movimentação continua a mesma para as duas classes. O que foi modificado foi o método de comparação entre objetos do tipo `Movemet` por questão de eficiência do sistema. Para o PRVFHJT usou-se uma fábrica especifica para produzir estes movimentos. Esta fábrica é implementada pela classe `VRPRealocationMovementFactory` e foi construída pelo fato de que a posição de índice zero da solução deve sempre ser ocupada por um veículo. Logo, movimentos que realoquem clientes para a posição zero não devem ser produzidos.

4.7 Penalidades

Foram estabelecidas três penalidades diferentes: uma para o caso da rota superar o limite de distância pré-estabelecido, uma para o caso da rota superar o limite de tempo também pré-estabelecido e uma terceira para o caso de demanda da rota superar, em algum momento, a capacidade do veículo.

4.8 Cálculo dos custos

A partir do cálculo da distância para as rotas, é calculado o custo para percorrer as mesmas. Em seguida, calcula-se os tempos das mesmas levando em conta o tempo de descarga, o tempo fixo de atendimento e a presença das janelas de tempo. Caso um veículo chegue a um cliente durante um intervalo entre janelas de tempo, o veículo deverá permanecer naquele cliente até que a próxima janela comece, atendendo, assim este cliente. As paradas são feitas à medida em que seu tempo de início é atingido, de modo que sua execução não inviabilize a realização da rota. Por exemplo, se uma rota já se encontra no tempo 50 e uma parada deveria ser realizada a partir do tempo 45 e esta parada levaria 40 minutos, suponha que a janela de tempo do cliente no qual o veículo se encontra agora, fecharia no tempo 60. Se a parada fosse realizada antes do atendimento, o veículo teria que esperar até a próxima janela de tempo. Deste modo, o atendimento deve ser feito primeiro. Logo após o atendimento, a parada será realizada.

Após calculados os custos, e estabelecidas as inviabilidades com relação aos limites, os custos totais são calculados por uma soma dos custos com uma multiplicação das inviabilidades por suas penalidades:

$$\text{Custo penalizado} = \sum_{i=1}^{3} inviabilidade_i \times penalidade_i$$

$$\text{Custos totais} = \sum_{i=1}^{3} custos_i$$

FO = custos totais + custo penalizado

5 Resultados obtidos

O *framework* e a aplicação discutidos neste trabalho foram implementado em Java e testados em um microcomputador Pentium IV 2,8 GHz HT, com 512 MB de RAM. Para fins de validação, a aplicação foi comparada com um aplicativo de roteirização disponível no mercado, chamado LogWare. O LogWare é um software comercial que usa uma heurística convencional, provavelmente da década de 60 e, portanto, chega sempre ao mesmo resultado. Isto não acontece com a aplicação proposta neste trabalho, a qual, por não ser determinística, pode chegar a soluções finais diferentes.

As instâncias usadas para testar a aplicação desenvolvida são as existentes no LogWare. A primeira, a qual diz respeito a um problema de roteamento com frota heterogênea sem janelas de tempo, tem um conjunto de 15 clientes e 4 veículos. A segunda se refere a um problema de roteamento com frota homogênea e janelas de tempo, com 21 clientes e 21 veículos.

A aplicação desenvolvida, denominada AP, foi testada hibridizando-se as metodologias *Simulated Annealing* e Busca Tabu. O método *Simulated Annealing* parte de uma temperatura inicial determinada por simulação e sua solução final é refinada pelo procedimento de Busca Tabu. Os parâmetros adotados foram: Número de iterações em uma dada temperatura igual a 500, Tamanho da lista tabu no intervalo [5,10], Modificação no tamanho da lista tabu a cada 10 iterações, e número máximo de iterações sem melhora igual a ??????????????????????.

Tendo em vista que a aplicação desenvolvida, denominada AP, é baseada em um procedimento metaheurístico e que, portanto, as soluções finais dependem da seqüência de números aleatórios usados, foram feitas 10 execuções da aplicação para cada instância. A Tabela 5.1 mostra os resultados obtidos relativos à primeira instância, que não considera janelas de tempo. Nesta tabela, a coluna AP-custo indica o valor final da função objetivo, AP-Tempo indica o tempo de execução em milisegundos. A coluna LogWare-custo indica o valor da função objetivo retornado pelo software LogWare e a coluna LogWare-tempo, o tempo gasto pelo Logware para resolver a instância.

Execução	AP - custo	AP – Tempo (ms)	LogWare - custo	LogWare – tempo (ms)
1	7844*	11203	8489	996
2	7844*	11219	8489	996
3	8112*	10796	8489	996
4	7944*	10797	8489	996
5	7844*	10937	8489	996
6	7844*	10984	8489	996
7	8112*	11094	8489	996
8	7965*	11000	8489	996
9	8037*	15672	8489	996
10	7985*	11203	8489	996

Tabela 5. 1 resultados comparativos para o problema sem janelas de tempo

Observa-se que, em relação à qualidade das soluções, a aplicação proposta em 100% dos casos produz uma solução melhor que a do Logware para a instância exemplo sem janelas de tempo.

A Tabela 5.2 mostra 10 execuções da aplicação proposta para a segunda instância, que considera janelas de tempo.

Execução	AP - Custo	AP – Tempo (ms)	LogWare - custo	LogWare – tempo (ms)
1	4235*	46391	5095	996
2	4254*	44797	5095	996
3	4171*	44782	5095	996
4	4374*	44890	5095	996
5	4223*	44547	5095	996
6	4293*	46203	5095	996
7	4263*	45422	5095	996
8	4263*	46985	5095	996
9	4235*	45937	5095	996
10	4268*	44609	5095	996

Tabela 5. 2 Resultados comparativos para o problema com janelas de tempo

Observa-se que, em relação à qualidade das soluções, o método proposto em 100% dos casos produz uma solução melhor que a do Logware para a instância teste com janelas de tempo.

A Tabela 5.3 faz um comparativo entre o resultado do LogWare e a média dos custos da aplicação desenvolvida.

Instância do Problema	Custo Médio AP	Custo Logware	% Melhora	Tempo Médio AP (s)	Tempo médio Logware (s)
Sem JT	7953,1	8489	6,4	11490,5	996
Com JT	4257,9	5095	16,5	45456,3	996

Tabela 5. 3 Comparativo das soluções finais do Logware e do aplicativo proposto

Obs :1 – o LogWare não possui mecanismo para medir o tempo de execução, então foi utilizado um relógio cronômetro para fazer a medição.

A Tabela 5.4 mostra 10 execuções da aplicação proposta para uma instância teste da literatura, a qual possui 50 clientes e 11 veículos e se refere a um problema de roteamento com frota heterogênea e sem janelas de tempo.

Execução	AP – Custo	AP – Tempo (ms)
1	2680	128750
2	2719	135094
3	2731	126187
4	2733	127063
5	2731	124078
6	2685	126672
7	2703	125156
8	2727	128469
9	2686	131141
10	2705	127343
Média	2710	127995,3

Tabela 5. 4 Execuções para uma instância da literatura

O melhor resultado conhecido na literatura para este problema é 2640. O valor médio das soluções produzidas pela metodologia proposta é de 2710. Portanto, o desvio entre esta média e o resultado da literatura é de apenas 2,7%.

Do ponto de vista da engenharia de software, o projeto mostrou-se bem sucedido, uma vez que a construção de um protótipo de *framework* para fins de otimização mostrou-se perfeitamente viável.

Várias classes foram inteiramente reutilizadas e muitas outras foram reutilizadas parcialmente através de especialização.

Para ilustrar o nível de reuso alcançado durante o projeto, fez-se a comparação da aplicação proposta, com um outro aplicativo que também trabalha na resolução do PRVFHJT, desenvolvida também pelo autor do presente trabalho, cujo detalhamento está disponível em http://www.decom.ufop.br/prof/marcone/Orientacoes/OrientacoesConcluidas.htm.

A Tabela 5.5 faz um comparativo entre a aplicação proposta neste projeto e o aplicativo construído sem o uso de técnicas de engenharia de software.

Software	Aplicação proposta	Aplicativo sem uso de técnicas de engenharia de software
Manutenção	Fácil	Difícil
Modificação	Fácil	Difícil
Expansão	Fácil	Difícil
Tamanho	115 K	137 K

Tabela 5. 5 Comparativo entre a aplicação construída e um outro aplicativo

A aplicação proposta torna-se de fácil manutenção, modificação e expansão devido à alta modularidade com a qual ela foi construída. O aplicativo sem o uso das técnicas de engenharia de software não possui as mesmas características por se tratar de um bloco monolítico.

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 estas modificações por todas as classes do sistema.

Em contrapartida, uma simples modificação, como a mudança do nome de uma variável no aplicativo construído sem técnicas de engenharia de software pode acarretar uma série de alterações em todo o código do aplicativo, dificultando sua manutenção, modificação e expansão.

Apesar de ser menor que o aplicativo anterior, a aplicação proposta neste trabalho é mais flexível do ponto de vista da manutenção, modificação e expansão. Além disso, o aplicativo usado para comparação possui apenas uma estrutura de vizinhança, a de troca, enquanto a aplicação proposta possui duas estruturas de vizinhança, a de troca e a de realocação.

O aplicativo anterior também é mais limitado do ponto de vista do processo de otimização, uma vez que ele só pode realizar o processo de otimização pelo método *Simulated Annealing*, enquanto a aplicação proposta pode fazer o processo de otimização pelos métodos *Simulated Annealing* e *Busca Tabu* separadamente ou em conjunto.

O LogWare mostrou-se incapaz de resolver satisfatoriamente esta instância de problema pois a solução apresentada por ele não inclui todos os clientes nas rotas de atendimento, violando um dos requisitos essenciais do problema.

6 Conclusões

Conclui-se, como é reportado na literatura, que a construção de um *framework* não é uma tarefa fácil. Um grande conhecimento do domínio para o qual se quer construir um *framework* é necessário, além de conhecimento em técnicas de engenharia de software.

Todo o trabalho para se construir um *framework* para fins de otimização, a partir dos resultados obtidos neste projeto, parece ter sido bem empregado, uma vez que a construção do JFFO mostrou-se possível e os resultados alcançados atendem as expectativas iniciais do projeto e mostram que este pode ser um caminho promissor para as áreas de otimização combinatória e pesquisa operacional.

A aplicação proposta para a validação do JFFO mostra que o objetivo inicial do projeto foi alcançado e também que através de métodos metaheurísticos foi possível resolver satisfatoriamente uma instância de um problema complexo da literatura. Também pôde-se confirmar através deste estudo, a afirmação de Cordeau [9], que diz que os softwares de mercado são baseados em metodologias ultrapassadas, uma vez que a metodologia proposta produziu resultados significativamente melhores que a do software LogWare.

Para trabalhos futuros sugere-se:

- Agregar novas funcionalidades ao JFFO, adicionando novos algoritmos de otimização, novas estruturas de soluções e novas estruturas de movimentação.
- Construir outras aplicações, tratando de problemas diferentes, para validação.
- Usar o JFFO para fazer testes de metaheurísticas que trabalhem em paralelo.

Referências Bibliográficas

[1] Costa Jr., A. C. T., O Problema de Roteamento Periódico de Veículos: Uma abordagem via Metaheurística GRASP. Niterói, Rio de Janeiro, 2003. Dissertação de mestrado em Ciência da Computação - Universidade Federal Fluminense (UFF).

[2] Golden, B.; Assad, A.; Levy, L. & Gheysens, F.. The fleet size and mix vehicle routing problem. *Computers & Operations Research*, vol. 11, pp. 49-66, 1984.

[3] Renaud, J. & Boctor, F. F.. Discrete Optimization, A sweep-based algorithm for the feet size and mix vehicle routing problem. *European Journal of Operational Research*, vol. 140, p. 618-628, 2002.

[4] Gendreau, M.; Laporte, G.; Musaraganyi, C. & Taillard, E. D.. A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, vol. 26, pp. 1153-1173, 1999.

[5] Souza, M. J. F., Notas de aula para métodos numéricos, Departamento de computação / iceb / ufop. Disponível em:
http://www.decom.ufop.br/prof/marcone/Disciplinas/MetodosNumericoseEstatisticos/QuadradosMinimos.pdf

[6] Novaes, A. G.. A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, vol. 26, pp. 1153-1173, 2002.

[7] Mine, O. M.. Abordagem Metaheurística para o Problema de Roteamento Periódico de Veículos. Ouro Preto, Minas Gerais, 2003. Relatório Técnico Final – DECOM, Universidade Federal de Ouro Preto (UFOP).

[8] Gendreau, M.; Laporte, G.; Potvin, J-Y. Metaheuristics for the Vehilcle Routing Problem. *Centre de Recherche sur les transports* – publication #963, 1994.

[9] Cordeau, J-F; Gendreau, M.; Laport, G.; Potvin, J-Y; Semet, F.. A guide to vehicle routing heuristics. *Journal of the Operational Research Society*, vol: 53, p.512-522, 2002.

[10] Tan, K.C.; Lee, L.H.; Zhu, Q.L.; Ou, K.. Heuristic Methods for Vehicle Routing Problem with Time Windows. *Artificial intelligence in Engineering*, vol. 15 pp. 281-295, 2001.

[11] Graccho, M.; Porto, S. C. S.; TabOOBuilder: An Object-Oriented Framework for Building Tabu Seach Aplications. Proceedings of the Third Metaheuristics International Conference, p. 247-253. Angra dos Reis, Julho de 1999.

[12] Souza, M. J. F., Notas de aula para Inteligência Computacional para Otimização, Departamento de computação / iceb / ufop. Disponível em:
http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.pdf

[13] Bräysy, O. & Gendreau, M.. Genetic Algorithms for the vehicle routing problem with time windows. Vaasa, Finlandia, 2001. Department of Mathematics and Statistcs, University of Vaasa.

[14] Breedam, A. V.. Improvement Heuristics For The Vehicle Routing Problem Based On Simulated Annealing. *European Journal Of Operational Research*, vol. 86, pp. 480-490, 1995.

[15] Liu, F.-H. & Shen, S.-Y.. A Metod For Vehicle Routin Problem With Multiple Veicle Types And Time Windows. Hsinchu, Taiwan, 1999. Department Of Industrial Engeneering And Management, National Chiao Tung University.