Learning and Nonlinear Models (L&NLM) - Journal of the Brazilian Society on Neural Networks, Vol. 8, Iss. 2, pp. 100-110, 2010

UM ALGORITMO EVOLUTIVO HÍBRIDO PARA O PROBLEMA DE RECOBRIMENTO DE ROTAS COM COLETA DE PRÊMIOS

Matheus de Souza Alves Silva, Marcio Tadayuki Mine, Luiz Satoru Ochi
Universidade Federal Fluminense – Niterói, Rio de Janeiro, Brasil
{msalves, mmine, satoru}@ic.uff.br

Marcone Jamilson Freitas Souza
Universidade Federal de Ouro Preto – Ouro Preto, Minas Gerais, Brasil
marcone@iceb.ufop.br

Resumo – Este artigo propõe um algoritmo evolutivo híbrido para obter soluções aproximadas para o Problema de Recobrimento de Rotas com Coleta de Prêmios (PRRCP). O algoritmo proposto combina estratégias heurísticas baseadas nos procedimentos Busca Local Iterada, Busca em Vizinhança Variável, Reconexão por Caminhos e GENIUS. Resultados computacionais para um conjunto de instâncias mostram a eficiência e a robustez da heurística proposta.

Palavras-chave – Inteligência Computacional, Metaheurísticas, Algoritmo Evolutivo.

Abstract – This paper proposes a hybrid evolutionary algorithm for getting an approximate solution for the Prize Collecting Covering Tour Problem (PCCTP). The proposed algorithm combines heuristic strategies based on Iterated Local Search, Variable Neighborhood Descent, Path Relinking and GENIUS procedures. Computational results on a set of instances illustrate the effectiveness and the robustness of the proposed heuristic.

Keywords – Computational Intelligence, Metaheuristics, Evolutionary Algorithm.

1. INTRODUÇÃO

Com o aumento da complexidade dos processos produtivos, há uma necessidade cada vez maior da utilização de sistemas inteligentes, que auxiliem o processo de tomada de decisão da melhor maneira possível. Métodos clássicos de otimização têm encontrado dificuldade para obter a melhor solução, dita ótima, mesmo quando alguns deles possuem teoricamente a garantia de atingí-la. A elevada complexidade dos problemas de otimização encontrados em diferentes áreas tem provocado a necessidade de desenvolvimento de novos métodos mais eficientes na prática para solucionar tais problemas. Esses métodos são usualmente o resultado da adaptação de conceitos de várias áreas. Um exemplo bem sucedido são as metaheurísticas, ou heurísticas inteligentes. A principal característica desta categoria de métodos é a possibilidade de encontrar diferentes ótimos locais durante a busca pela melhor solução. Entre esses métodos, destacam-se os Algoritmos Evolutivos, os quais têm se mostrado eficientes na resolução de vários problemas combinatórios.

Neste trabalho, desenvolvemos um Algoritmo Evolutivo Híbrido (AEH) para resolver o Problema de Recobrimento de Rotas com Coleta de Prêmios (PRRCP). O AEH combina estratégias heurísticas baseadas nos procedimentos heurísticos *Iterated Local Search* [1, 2], Busca em Vizinhança Variável [3–5], GENIUS [6] e Reconexão por Caminhos [7].

O PRRCP, referido na literatura como *Prize Collecting Covering Tour Problem* (PCCTP), é uma variante do Problema de Recobrimento de Rotas (PRR), o qual, por sua vez, é uma generalização do Problema do Caixeiro Viajante (PCV).

O PRR pode ser definido em um grafo não-direcionado $G = (N, E)$, sendo $N = V \cup W$, com $V = T \cup (V \backslash T)$ representando o conjunto dos vértices que podem fazer parte da rota (solução), W o conjunto dos vértices que precisam ser cobertos, $E = \{(v_i, v_j) \mid v_i, v_j \in V\}$, T o conjunto dos vértices obrigatórios, isto é, que *devem* fazer parte da rota e $V \backslash T$, o conjunto dos vértices *opcionais*, que não necessariamente precisam fazer parte da rota. Diz-se que um vértice $w_j \in W$ está coberto quando existe na solução pelo menos um vértice $v_i \in V$ tal que $d(w_j, v_i) \leq D$, sendo D um parâmetro do problema e d uma função real que retorna a distância entre os vértices w_j e v_i. No PRR o objetivo é determinar uma rota de comprimento mínimo sobre um subconjunto de V, contendo todos os vértices obrigatórios T e cobrindo todos os vértices de W.

Uma vez que o PRR pode ser reduzido ao PCV fazendo-se $D = 0$, $W = \emptyset$ e $N = T$, e este se enquadra na classe de problemas NP-difíceis, então o PRR também o é.

O PRRCP, além das restrições comuns ao PRR, possui as seguintes particularidades: (*i*) a cada vértice i de V está associado um prêmio não-negativo p_i; (*ii*) a quantidade dos prêmios coletados nos vértices presentes na solução tem que ser maior ou igual à quantidade mínima pré-estabelecida, dada por *PRIZE*. O objetivo do PRRCP, assim como no PRR, é encontrar uma rota de comprimento mínimo em um subconjunto de V, satisfazendo as restrições de pertinência de todos os vértices de T à solução, a cobertura de todos os vértices de W e por fim, a coleta da quantidade mínima de prêmios (*PRIZE*).

O PRRCP tem uma aplicação possível na realização dos "Serviços de Atendimento Médico Móvel" (SAMM), que funciona da seguinte forma: um veículo sai de um ponto origem, visita um conjunto de pontos de atendimento (vértices de T e eventualmente alguns vértices do conjunto $V \setminus T$), de forma que a população de uma determinada região (conjunto W) seja contemplada com tais serviços sem que nenhuma pessoa tenha que se locomover mais que uma distância máxima D para chegar a um ponto de parada do veículo. Ao final, o veículo retorna ao local de origem. Os pontos de parada do veículo são definidos a partir de valores como: i) pontos de parada pré-estabelecidos (vértices de T), ii) pontos onde a quantidade de pessoas residentes na proximidade seja alta (prêmio p_i do vértice $i \in V$), e/ou que não estejam cobertas por pontos de T (vértices de $V \setminus T$).

Analogamente ao PRR, o PRRCP também é considerado um problema NP-difícil, uma vez que pode ser reduzido ao PCV quando $D = 0$, $N = T$, *PRIZE* = 0 e $W = \emptyset$.

Outra variante do PRR fortemente relacionada ao PRRCP é o Problema de Recobrimento de Rota Generalizado (PRRG). Esse problema consiste em determinar uma rota de comprimento mínimo em um subconjunto de vértices $V \cup W$, permitindo que os vértices de W façam parte da rota solução, ao contrário do PRR, cobrindo a si próprio e a outros vértices deste conjunto, quando for o caso.

Dada a dificuldade de obtenção de soluções ótimas do PRRCP por métodos de programação matemática, em tempos computacionais aceitáveis para os casos de interesse prático, é desafiador o desenvolvimento de algoritmos eficientes para resolvê-lo. O algoritmo evolutivo híbrido proposto é uma contribuição do presente trabalho ao estudo deste problema para encontrar soluções sub-ótimas de qualidade.

O restante deste trabalho está organizado como segue. Na Seção 2 é feita uma breve revisão dos trabalhos relacionados. O detalhamento da metodologia proposta para resolver o PRRCP é apresentação na Seção 3. Na Seção 4 são apresentados e analisados os resultados obtidos com a aplicação do algoritmo proposto. Por fim, na Seção 5 são apresentadas as conclusões.

2 TRABALHOS RELACIONADOS

Apesar de possuir várias aplicações possíveis, o PRR não tem sido muito estudado pelos pesquisadores, desde que foi introduzido, em 1981, na tese de doutorado de Current [8]. Em [9] e [10] foi desenvolvida uma heurística que gera um conjunto de soluções para PRR em duas etapas. Na primeira, minimiza-se o comprimento da rota. Na segunda, maximiza-se o número de vértices cobertos pela rota gerada na primeira etapa.

Posteriormente, em [11] foi proposto um procedimento baseado na heurística GENIUS [6] e no Algoritmo *PRIMAL Set Covering* [12] para resolução do PRR. Além desse procedimento, os autores propuseram quatro regras de redução, um algoritmo exato baseado na técnica *Branch-and-Cut* e uma formulação de programação matemática.

Em [13] os autores apresentaram uma nova formulação matemática para o PRR. Além disso, também apresentaram heurísticas *Scatter Search* para o PRR. Em [14] foi proposta uma nova formulação matemática e uma heurística GRASP para resolver o PRR.

Para o PRRG, é de nosso conhecimento somente os trabalhos [14] e [15], nos quais são propostas uma formulação matemática, um conjunto de regras de redução e métodos heurísticos para sua resolução.

Por ser um problema relativamente novo, existem poucos trabalhos na literatura relacionados ao PRRCP. Em [16] são propostas uma formulação matemática, uma regra de redução dos vértices do problema e uma heurística GRASP [17]. Em [18], base do presente trabalho, são propostas três novas regras de redução para o PRRCP, uma nova formulação de programação matemática e duas abordagens heurísticas baseadas em *Iterated Local Search*.

3 METODOLOGIA

3.1 REPRESENTAÇÃO DE UMA SOLUÇÃO PARA O PRRCP

Uma rota, ou solução s do problema, é representada por um vetor contendo no máximo $|T| + |V \setminus T|$ posições, pois como no PRRCP apenas um subconjunto dos vértices compõe a solução, o tamanho desse vetor pode variar de uma solução para outra.

3.2 SOLUÇÃO INICIAL

Para gerar uma solução inicial são utilizados cinco algoritmos construtivos: 1) Os algoritmos *ADD* e *DROP* de [16]; 2) Adaptação ao PRRCP dos algoritmos de Inserção Mais Barata e do Vizinho Mais Próximo e 3) Adaptação da heurística GENIUS [6]. Todos esses algoritmos são adaptados para uma versão construtiva parcialmente gulosa, como na fase de construção do algoritmo GRASP [17], adotando-se um parâmetro α para definir o nível de aleatoriedade do procedimento construtivo. O detalhamento desses algoritmos pode ser encontrado em [18].

3.3 ESTRUTURAS DE VIZINHANÇA

Para explorar o espaço de soluções do PRRCP, utilizam-se 17 estruturas de vizinhança, formadas a partir de nove tipos de movimentos simples e oito compostos.

Os cinco primeiros movimentos simples são tradicionais em heurísticas de refinamento do PCV, a saber: (a) *Shift* - consiste em remover um vértice de uma sequência e reinserí-lo em outra posição; (b) *Swap* - consiste em trocar dois vértices de suas posições da sequência; (c) *Or-Opt* - extensão do movimento *shift*, consistindo em transferir n vértices consecutivos da sequência para uma outra posição na rota, com $n \in [2, |V| - 2]$; (d) 2-*Opt* - consiste em remover duas arestas da solução e inserir duas

novas arestas de forma que continue existindo apenas um ciclo; (e) 3-*Opt* - consiste na remoção de três arestas da solução e inserção de três novas arestas de forma que continue existindo apenas um ciclo. Diferentemente do movimento 2-*opt*, em que a inserção é única, no 3-*Opt* há quatro maneiras de se inserir mantendo-se um ciclo.

Além desses movimentos, são usados três movimentos baseados nas fases de inserção e remoção da heurística GENIUS [6] (a saber, *addGenius*, *dropGenius* e *addCheapestInsertion*) e no movimento *dropSimple* de inserção do algoritmo de Inserção Mais Barata. O movimento *addGenius* consiste em inserir um vértice na solução por meio dos dois tipos de inserção da fase GENI da heurística GENIUS. Já o movimento *dropGenius* remove um vértice da solução utilizando os dois tipos de remoção da fase US da heurística GENIUS. O movimento *cheapestInsertion* procura a melhor posição entre dois vértices adjacentes da solução para inserir um vértice. A posição que trouxer o menor custo adicional entre todos os pares de vértices adjacentes é escolhida. O último movimento simples, *dropSimple*, consiste em remover um vértice da solução e inserir uma aresta ligando os dois vértices anteriormente adjacentes ao vértice removido.

Uma característica do PRRCP é que nem todos os vértices de $V \setminus T$ são necessários à solução, alguns por motivos claros como, por exemplo, pelo fato de não cobrirem vértices de W e o prêmio mínimo já estiver sido satisfeito. Por outro lado, pode existir dúvida em determinar quais vértices de $V \setminus T$ farão parte da solução, pois a utilização de um determinado vértice deste conjunto pode ser melhor que a de outro desse mesmo conjunto quanto à utilização de ambos na solução. Para solucionar tal questão, é preciso testar diferentes inserções e remoções de vértices que não fazem parte da solução corrente, como também remover e reinserir um mesmo vértice, pois pode ser que ele não se encontre em sua posição ideal. Diante disso, considerou-se no presente trabalho oito combinações entre os quatro últimos movimentos desenvolvidos, de forma a permitir inserir e remover novos vértices na solução e reinserir um vértice na tentativa de melhorar e muitas vezes viabilizar a solução. Foram estes os movimentos compostos:

- *dropGenius_addGenius*: retira um vértice utilizando a fase de remoção US e insere um novo vértice utilizando a fase de inserção GENI, ambos da heurística GENIUS;
- *dropGenius_re_addGenius*: retira um vértice utilizando a fase de remoção US e o reinsere utilizando a fase de inserção GENI, ambos da heurística GENIUS;
- *dropGenius_addCheapestInsertion*: retira um vértice utilizando a fase de remoção US da heurística GENIUS e insere um novo vértice com base na heurística da Inserção Mais Barata;
- *dropGenius_re_addCheapestInsertion*: retira um vértice utilizando a fase de remoção US da heurística GENIUS e o reinsere utilizando com base na Inserção Mais Barata;
- *dropSimple_addGenius*: retira um vértice apenas religando as arestas dos vértices adjacentes ao vértice removido e insere um novo vértice utilizando a fase de inserção GENI da heurística GENIUS;
- *dropSimple_re_addGenius*: retira um vértice apenas religando as arestas dos vértices adjacentes ao vértice removido e o reinsere utilizando a fase de inserção GENI da heurística GENIUS;
- *dropSimple_addCheapestInsertion*: retira um vértice apenas religando as arestas dos vértices adjacentes ao vértice removido e insere um novo vértice aplicando-se a heurística da Inserção Mais Barata;
- *dropSimple_re_addCheapestInsertion*: retira um vértice apenas religando as arestas dos vértices adjacentes ao vértice removido e o reinsere pela heurística da Inserção Mais Barata.

Cada movimento simples e/ou composto realizado a partir de uma solução s define uma estrutura de vizinhança $N^{\,k}(s)$, com $k = 1, \cdots, 17$.

3.4 AVALIAÇÃO DE UMA SOLUÇÃO

Para avaliar uma solução s, utiliza-se uma função $f(s)$ definida pela Equação (1). Esta função consiste em somar todos os custos, no caso distâncias, de todas as arestas presentes na solução e penalizar o não atendimento das restrições do problema.

$$f(s) = \sum_{i,j \in s} dist_{ij} + \sum_{k \in C} \mu_k \times inv_k \tag{1}$$

em que:

- s : solução que está sendo avaliada;
- $f(s)$: função de avaliação;
- C : conjunto de restrições do problema;
- $dist_{ij}$: distância de um vértice i a um vértice j, com $i, j \in s$;
- μ_k : penalidade por desrespeitar a restrição $k \in C$;
- inv_k : número de vezes em que a restrição k é desrespeitada.

3.5 ALGORITMO EVOLUTIVO HÍBRIDO PROPOSTO

O termo Algoritmo Evolutivo (AE) compreende uma família de resolvedores de problemas com base em princípios que podem ser encontrados na evolução biológica [19].

As três linhas principais de estudo nessa área são o Algoritmo Genético (AG), Estratégias de Evolução (EE) e Programação Evolutiva (PE).

Um AE típico funciona da seguinte forma: a cada geração constrói-se um conjunto de indivíduos, representando as soluções de um problema de otimização. Esses indivíduos, juntamente com a população vinda de gerações anteriores, são combinados numa fase de cooperação para formarem novos indivíduos. Esses novos indivíduos passam, então, por uma fase de adaptação antes de se decidir quais serão incluídos na população que irá para a próxima geração. O algoritmo finaliza ao se atingir um número máximo de gerações [20].

No Algoritmo Evolutivo proposto para resolver o PRRCP, denotado por AEH, utiliza-se o conceito de classes para diferenciar os indivíduos. A classe A, representada pelo *pool* de soluções elite, contém os melhores indivíduos, em termos de custo da solução, encontrados durante as gerações. Esses indivíduos são substituídos somente se melhores indivíduos forem criados. A classe B é representada por uma parcela da população que também só é substituída caso indivíduos de melhor custo sejam gerados.

Por fim, a classe C é representada pela parcela restante da população com indivíduos de custos mais elevados, substituídos em todas as gerações por novo processo construtivo.

Para gerar a população inicial, utilizam-se versões randomizadas dos cinco métodos construtivos apresentados na Seção 3. Como tais métodos possuem ideias de construção diferentes, escolhendo-se diferentes procedimentos de construção, geram-se possivelmente diferentes indivíduos, promovendo assim uma miscigenação destes.

O AEH também faz uso de procedimentos de refinamento baseados no Método de Descida Randômica – MDR (descrito em detalhes na Subseção 3.6), ILS-VNRD e VNRD (detalhados nas subseções 3.8 e 3.9, respectivamente), bem como do procedimento de Reconexão por Caminhos (descrito na Subseção 3.7) entre soluções das classes B e C em direção à soluções do *pool*.

O Algoritmo 1 apresenta o pseudocódigo do Algoritmo Evolutivo Híbrido proposto. Neste Algoritmo, $ger_{\max}$ representa o número máximo de gerações do algoritmo AEH, *populationSize* indica o tamanho da população, *classBSize* o número de indivíduos da classe B, *classCSize* o número de indivíduos da classe C, *eliteSetSize* o tamanho do *pool* de soluções elite, *percDiffSolution* a porcentagem mínima de diferença de uma solução para as demais soluções do *pool*, $iterMDR_{\max}$ o número máximo de iterações sem melhora do método MDR, $iterVNRD_{\max}$ o número máximo de iterações do método VNRD, $kp_{\min}$ e $kp_{\max}$ os números mínimo e máximo assumidos por *kp*, respectivamente, e δ a variação permitida para *kp*.

Algoritmo 1: AEH

Entrada: $ger_{\max}$, *populationSize*, *classBSize*, *classCSize*, *eliteSetSize*, *percDiffSolution*, $iterMDR_{\max}$, $iterVNRD_{\max}$, $kp_{\min}$, $kp_{\max}$, δ, α
Saída: s

```
1  início
2      enquanto população e pool não completos faça
3          s_0 ← SolucaoInicial(α)          // Escolha, aleatoriamente, um método da Subseção 3.2
4          s ← MDR(s_0)                     // Refine a solução pelo método MDR da Subseção 3.6
5          Atualize pool
6          Atualize população(classes B e C)
7      fim
8      ger ← 1
9      enquanto ger ≤ ger_max faça
10         s_base ← s ∈ população(classes B e C)
11         s_guia ← s ∈ pool
12         s_RC ← ReconexaoPorCaminhos(s_base, s_guia)          // Aplique a estratégia da Subseção 3.7
13         s ← ILS-VNRD(s_RC)          // Refine a solução pelo método ILS-VNRD da Subseção 3.8
14         Atualize pool
15         ger ← ger + 1
16         para classCSize iterações, dada pelo número de indivíduos da classe C faça
17             s_0 ← SolucaoInicial(α)      // Aplique um dos métodos construtivos da Subseção 3.2
18             s ← MDR(s_0)                 // Refine a solução pelo método MDR da Subseção 3.6
19             Atualize população(classes B e C)
20         fim
21     fim
22     s ← s* ∈ pool                                  // Escolha a melhor das soluções do pool
23     retorne s
24 fim
```

Neste algoritmo, inicialmente são criados vários indivíduos para compor uma populacao e o *pool* de soluções elite. Esses indivíduos são formados após escolha aleatório de um dos métodos construtivos apresentados na Subseção 3.2. Cada indivíduo é, então, submetido a uma busca local realizada pelo Método Randômico de Descida - MDR (descrito na Subseção 3.6). Se esse indivíduo s for melhor que o indivíduo que possuir o pior custo dentre todas as soluções do *pool* e possuir um percentual mínimo de diferença para todas as soluções desse conjunto elite, então o *pool* é atualizado com a inclusão de s. Caso esse indivíduo não satisfaça aos requisitos para entrar no *pool* (linha 5 do Algoritmo 1), verifica-se se s pode fazer parte da população. Para isso é preciso que ele seja melhor que a pior solução desse conjunto, não precisando ter um percentual mínimo de diferença em relação aos demais indivíduos. Criada a populacao inicial, enquanto o número máximo de geração ($ger_{\max}$) não for alcançado, aleatoriamente escolhe-se uma solução da populacao para ser a solução base do procedimento Reconexão por Caminhos (linha 12 do Algoritmo 1), detalhado mais adiante na Subseção 3.7. A solução guia é selecionada do *pool* de soluções elite de acordo com o grau de diferença em relação à solução base. A solução que possuir o maior grau de diferença será escolhida como guia. A melhor solução encontrada durante a Reconexão por Caminhos é então submetida a uma busca local realizada pelo procedimento ILS-VNRD descrito na Subseção 3.8 (linha 13 do Algoritmo 1). Caso a solução s retornada pelo ILS-VNRD seja melhor que a solução de maior custo do *pool* e se s possuir uma porcentagem mínima de diferença para as demais soluções do conjunto elite, dada por *percDiffSolution*, atualiza-se o *pool* com s. Ao final de cada geração reconstrói-se a classe C da populacao, substituindo-se seus indivíduos por novos indivíduos (linhas 17 a 19 do Algoritmo 1). Como a populacao é ordenada pelo custo dos indivíduos, pode acontecer de um novo indivíduo criado para a classe C possuir um custo menor que um indivíduo da classe B. Quando isto ocorre, realiza-se a troca de classes entre os indivíduos, isto é, automaticamente o novo indivíduo passa a ser da classe B e o outro indivíduo que pertencia a esta classe passa agora a pertencer à classe C. Ao final do algoritmo, retorna-se o melhor indivíduo do *pool* (linha 22 do Algoritmo 1).

3.6 MÉTODO DE DESCIDA RANDÔMICA

O Método de Descida Randômica, ou MDR, é uma heurística de refinamento que consiste em analisar um vizinho qualquer de uma dada solução e o aceitar somente se ele for estritamente melhor que a solução corrente. Caso esse vizinho não seja melhor, a solução corrente permanece inalterada e outro vizinho aleatório é gerado. O procedimento é finalizado quando se atinge um número máximo de iterações (dado por $iterMDR_{\max}$) sem que haja melhoria no valor da melhor solução obtida. Na adaptação feita, a cada iteração escolhe-se uma vizinhança N^k qualquer dentre as 17 desenvolvidas e, a seguir, um vizinho qualquer nessa vizinhança.

3.7 RECONEXÃO POR CAMINHOS

O procedimento de Reconexão de Caminhos [7] foi utilizado como fase de intensificação para o AEH proposto. Este algoritmo inicia calculando a diferença simétrica de todas as soluções do *pool* de soluções elite em relação à solução base, s_{base}, proveniente da populacao. A diferença simétrica pode ser entendida como a quantidade de passos que são necessários para sair de uma solução base e chegar a uma solução guia. Isto significa que a cada iteração, um atributo que exista na solução guia, mas que não pertença à solução base, é inserido nesta com o objetivo de que ao final do procedimento a solução base coincida com a solução guia.

Definida a solução base, s_{base}, enquanto ela possuir alguma diferença para s_{guia}, o procedimento aplica um movimento que consiste em inserir na sbase um atributo de s_{guia}. O movimento escolhido é o melhor dentre os possíveis movimentos que podem ser aplicados, isto é, o que trouxer o maior benefício para a solução. A essa solução intermediária, também chamada de candidata (s_{cand}), aplica-se uma busca local por meio do MDR. Se essa solução s for melhor que a pior solução do *pool* e atender ao critério de diversificação de soluções, então se atualiza o *pool* com s. O procedimento termina quando ocorre a convergência das soluções, isto é, quando as soluções base e guia possuem os mesmos vértices e a ordem de visita dos vértices em ambas as soluções é a mesma.

A expressão para o cálculo da diferença simétrica entre duas soluções é dada por:

$$DS\left(s_{guia}, s_{base}\right) = \frac{a+v}{|A|+|V|} \tag{2}$$

em que:

- a representa o número de arestas semelhantes das soluções;
- v representa o número de vértices semelhantes das soluções;
- $A = \left(A^{guia} \cup A^{base}\right) \setminus \left(A^{guia} \cap A^{base}\right)$ representa o número de arestas distintas entre as soluções;
- $V = \left(V^{guia} \cup V^{base}\right) \setminus \left(V^{guia} \cap V^{base}\right)$ representa o número de vértices distintos entre as soluções.

3.8 ILS-VNRD

O procedimento de busca local ILS-VNRD, descrito pelo Algoritmo 2, combina os procedimentos heurísticos ILS [1, 2], MDR e VNRD [3, 5]. Ele tem como solução inicial a melhor solução encontrada durante a Reconexão por Caminhos (vide

Subseção 3.7). Em seguida, aplica-se um procedimento de busca local, o VNRD (descrito na Subseção 3.8). A cada iteração do ILS-VNRD, gera-se uma perturbação na solução corrente. Essa perturbação consiste em realizar kp movimentos aleatórios dentre aqueles descritos na Subseção 3.3, sendo kp um número compreendido entre dois valores parametrizados, $kp_{\min}$ e $kp_{\max}$, conforme mostra o algoritmo a seguir.

Algoritmo 2: ILS-VNRD

Entrada: s_{RC}, $iterMDR_{\max}$, $iterILS_{\max}$, $iterVNRD_{\max}$, $kp_{\min}$, $kp_{\max}$, δ
Saída: s

```
1  início
2      s ← MDR(s_RC, iterMDR_max)
3      kp ← kp_min
4      iter ← 1
5      enquanto kp ≤ kp_max faça
6          enquanto iter − melhorIter ≤ iterILS_max faça
7              iter ← iter + 1
8              s' ← perturbacao(s, kp)
9              s" ← VNRD(s', iterVNRD_max)
10             se f(s") ≤ f(s) então
11                 s ← s"
12                 melhorIter ← iter
13                 kp ← kp_min
14             fim
15         fim
16         kp ← kp + δ
17     fim
18     retorne s
19 fim
```

À solução perturbada (linha 8 do Algoritmo 2) aplica-se o VNRD, tentando com isso gerar uma solução melhorada. Se essa for melhor que a solução corrente, então a solução melhorada é aceita como a nova solução corrente e reinicia-se o valor de kp. Essa repetição é executada até que um número máximo de iterações sem melhora seja realizado. Quando isso ocorre, incrementa-se o grau de perturbação, kp, em um fator δ, até que kp atinja seu valor máximo ($kp_{\max}$). A variação do grau de perturbação é uma estratégia importante, pois permite a intensificação da busca e diversificação das soluções geradas. A busca é intensificada quando o grau de perturbação é baixo e a diversificação ocorre quando esse grau atinge um valor relativamente alto.

3.9 DESCIDA RANDÔMICA EM VIZINHANÇA VARIÁVEL

O Método de Descida Randômica em Vizinhança Variável (*Variable Neighboohoord Randon Descent*, VNRD), é uma variante do Método de Descida em Vizinhança Variável (*Variable Neighboohoord Descent*, VND) [3, 5].

O VND é um método de refinamento que consiste em explorar o espaço de soluções por meio de trocas sistemáticas de estruturas de vizinhança (descritas na Seção 3). O método aceita somente soluções de melhora da solução corrente e retorna à primeira vizinhança quando uma solução melhor é encontrada.

A grande diferença do VNRD para o VND é que, no primeiro método, a exploração de cada vizinhança não é feita por completa, apenas um número limitado de vizinhos dela é explorado. No procedimento desenvolvido, são feitas *iterVNRD* $_{\max}$ iterações sem melhora na vizinhança corrente. O VNRD é finalizado quando nenhuma solução de melhora é encontrada após a aplicação de todas as estruturas de vizinhança utilizadas. O Algoritmo 3 mostra o pseudocódigo do método de busca local VNRD desenvolvido.

No Algoritmo 3, as r estruturas de vizinhança, nomeadas de N^1 a N^{17}, são aquelas descritas na Subseção 3.3, consideradas na seguinte ordem de exploração do espaço de busca:

- N^1: *addCheapestInsertion*;
- N^2: *addGENIUS*;
- N^3: *shift*;
- N^4: *swap*;
- N^5: *Or-opt*;
- N^6: 2-*opt*;
- N^7: 3-*opt*;

- N^8: *dropSimple_re_addCheapestInsertio*;
- N^9: *dropSimple_re_addGenius*;
- N^{10}: *dropGenius_re_addCheapestInsertion*;
- N^{11}: *dropGenius_re_addGenius*;
- N^{12}: *dropSimple_addCheapestInsertion*;
- N^{13}: *dropSimple_addGenius*;
- N^{14}: *dropGenius_addCheapestInsertion*;
- N^{15}: *dropGenius_addGenius*;
- N^{16}: *dropSimple*;
- N^{17}: *dropGenius*;

Algoritmo 3: VNRD

Entrada: s_0, *iterVNRD*$_{\max}$
Saída: s
início
  $s \leftarrow s_0$
  Sejam as r estruturas de vizinhança N^1 a N^{17} definidas na Subseção 3.3
  $k \leftarrow 1$
  $s^\star \leftarrow s$
  enquanto $k \leq r$ **faça**
    iter $\leftarrow 1$
    enquanto *iter* $\leq$ *iterVNRD*$_{\max}$ **faça**
      iter $\leftarrow$ *iter* $+ 1$
      Gere aleatoriamente um vizinho $s' \in N^k(s)$
      se $f(s') \leq f(s)$ **então**
        $s \leftarrow s'$
        iter $\leftarrow 1$
      fim
    fim
    se $f(s) < f(s^\star)$ **então**
      $s^\star \leftarrow s$
      $k \leftarrow 1$
    fim
    senão
      $k \leftarrow k + 1$
    fim
  fim
  $s \leftarrow s^\star$
  retorne s
fim

4 RESULTADOS E ANÁLISE

O Algoritmo Evolutivo Híbirdo (AEH) foi implementado na linguagem C++ utilizando o ambiente Microsoft Visual Studio 2008 e testado em um computador Intel Core 2 Duo, com 2.2 GHz e 2.5 GB de memória principal, rodando o sistema operacional Windows Vista. Para validá-lo, foram utilizados dois grupos de problemas-teste adaptados para o PRRCP a partir do repositório TSPLIB [21], a biblioteca de problemas-teste mais conhecida para o PCV. A justificativa para não se ter utilizado os problemas-teste do trabalho de [16] se deve à não disponibilidade de tais problemas na literatura.

Basicamente, a adaptação consiste em, a partir da matriz de distâncias dos vértices, das porcentagens de vértices que se deseja ter em cada um dos três conjuntos, encontrar uma distância de cobertura de forma que se consiga ter a porcentagem desejada de vértices de W com pelo menos algum vértice de V cobrindo esses vértices. Além disso, foram tomados outros cuidados, como por exemplo, de não permitir que apenas os vértices de T consigam cubrir os vértices de W e nem permitir que o prêmio mínimo a ser coletado possa ser satisfeito apenas com os vértices de T, pois caso esses casos aconteçam, descaracteriza-se o PRRCP, que se reduziria a um PCV, mas com apenas os vértices obrigatórios.

Os problemas-teste do PRRCP envolvem coleta mínima de 25%, 50% e 75% do prêmio total dos vértices de V. Com isso, em grande parte dos problemas, alguns vértices do conjunto $V \backslash T$ devem estar obrigatoriamente presentes na solução, o que dificulta sua resolução, uma vez que o número de combinações aumenta exponencialmente.

Os problemas-teste utilizados foram agrupados em dois conjuntos. No Grupo 1, o número de vértices varia entre 50 e 200; enquanto no Grupo 2 esse número varia de 201 a 400. O detalhamento da construção de tais problemas pode ser encontrado em [18].

Para cada problema-teste foram realizadas 10 execuções, cada qual partindo de uma semente diferente de números aleatórios. Os parâmetros adotados no Algoritmo Evolutivo Híbrido foram os seguintes: $ger_{\max}$ = 7, *populationSize* = 5, *classBSize* = 3, *classCSize* = 2, *eliteSetSize* = 5, *percDiffSolution* = 0,15, $iterMDR_{\max} = 300$, $iterILS_{\max} = 100$, $iterVNRD_{\max} = 100$, $kp_{\min}$ = 5, $kp_{\max} = 7$, $\delta = 2$ e $\alpha = 0,8$. Tais valores foram definidos após uma bateria preliminar de testes.

Para legitimar o desempenho do AEH proposto, este foi comparado com as duas melhores versões, descritas em [18], de um algoritmo de busca local baseado na metaheurística *Iterated Local Search*, tendo como método de busca local o MDR. Para o Grupo 1, o AEH foi comparado com a versão ILS-MDR-AD, a qual tem o procedimento ADD como método de geração de solução inicial. Para o Grupo 2, o AEH foi comparado com a versão ILS-MDR-IB, a qual usa o procedimento da Inserção Mais Barata para construir uma solução.

Esses algoritmos foram comparados pelos seguintes critérios: (a) custo médio de todos os problemas do grupo; (b) desvio médio das soluções em relação ao melhor custo conhecido; (c) taxa de sucesso de cada algoritmo em alcançar o melhor custo em pelo menos uma das dez execuções; (d) porcentagem de problemas que cada algoritmo encontrou o menor custo sozinho.

As Tabelas 1 e 2 resumem os resultados das comparações envolvendo problemas-teste do Grupo 1 e Grupo 2, respectivamente. Nessas tabelas, a primeira coluna indica o nome do algoritmo; a segunda, o custo médio de todos os problemas; a terceira, o desvio médio do custo das soluções em relação ao melhor custo conhecido e a quarta, o tempo médio gasto em cada execução do algoritmo. Na penúltima coluna mostra-se a taxa de sucesso de cada algoritmo em alcançar o melhor custo em pelo menos uma das dez execuções. Por fim, na última coluna, apresenta-se a porcentagem de problemas que cada algoritmo encontrou o menor custo sozinho.

O desvio médio é calculado pela Equação (3):

$$Desvio = \frac{Média - Melhor\ Valor}{Melhor\ Valor} \times 100 \tag{3}$$

Tabela 1: Comparação entre os algoritmos ILS-MRD-AD e AEH nos problemas-teste do **Grupo 1**

Versão	Custo	Desvio (%)	Tempo (s)	Sucesso (%)	Sozinho (%)
ILS-MRD-AD	20080	0,06	1255,9	96,4	0
AEH	20073,5	0,02	1538,3	99,1	1,8

De acordo com a Tabela 1, observa-se que o Algoritmo Evolutivo Híbrido (AEH) obteve um desempenho superior ao de ILS-MRD-AD. Isto se deve ao fato de que o desvio médio das soluções de AEH foi três vezes menor que o do algoritmo ILS-MRD-AD, no caso, de 0,02% e, além disso, o AEH encontrou o melhor valor conhecido em 99,1% dos casos do Grupo 1, e em 1,8% dos problemas-teste encontrou esse melhor valor sozinho.

Em relação aos tempos de execução, verifica-se que estes mostraram-se próximos, apesar de o Algoritmo Evolutivo possuir o procedimento Reconexão por Caminhos adicional em relação às outras propostas.

O motivo de a Reconexão por Caminhos não ter elevado muito o tempo de execução do Algoritmo Evolutivo é justificado pelo fato de o número de vértices nos problemas-teste do Grupo 1 não ser muito elevado e, dessa forma, pequenos esforços do algoritmo já melhoram bastante a qualidade da solução. No Algoritmo Evolutivo, a população é gerada pelos diferentes métodos de geração da solução inicial; com isso, os custos e, consequentemente, o arranjo dos vértices nas soluções é o mais variado possível. Com uma leve busca local realizada em cada indivíduo da população após sua criação, as soluções são bem melhoradas, formando-se uma população com uma qualidade melhor. Então, quando essas soluções são encaminhadas para a Reconexão por Caminhos, poucos movimentos precisam ser efetuados para se sair da solução base e alcançar a solução guia, o que faz com que o tempo de execução do algoritmo não cresça muito.

Tabela 2: Comparação entre os algoritmos ILS-MRD-IB e AEH nos problemas-teste do **Grupo 2**

Versão	Custo	Desvio (%)	Tempo (s)	Sucesso (%)	Sozinho (%)
ILS-MRD-IB	33752,7	0,34	4897,5	76,7	3.3
AEH	33641,2	0,14	9294	91,0	16,7

Com base na Tabela 2, observa-se que a diferença do desempenho do Algoritmo Evolutivo em relação ao ILS-MRD-IB foi mais significativa para os problemas-teste do Grupo 2. De fato, o AEH apresentou um desvio médio de 0,14%, menor que o do

algoritmo ILS-MRD-IB, que foi de 0,34%. Além disso, o número de problemas-teste que o Algoritmo Evolutivo encontrou o melhor valor conhecido, seja esse valor o ótimo ou um limite superior, foi superior a 90%, valor esse bem maior do que o alcançado pelo algoritmo ILS-MRD-IB, de 76,7%. Para os problemas-teste do Grupo 2, o AEH encontrou, sozinho, a melhor solução conhecida para aproximadamente 16,7% dos casos. Assim, tal como no caso anterior, também pode-se concluir que o AEH não necessita de soluções iniciais de boa qualidade para convergir para boas soluções finais.

Em relação ao tempo de execução, verifica-se que o AEH gastou em média o dobro de tempo que a outra abordagem proposta. Tal fato é justificado, ao contrário dos problemas-teste do Grupo 1, pela utilização da Reconexão por Caminhos. Nos problemas do Grupo 2, o conjunto de vértices é muito maior; com isso, a busca local realizada após a criação de cada indivíduo não melhora tanto a qualidade da solução como acontece nos problemas-teste do Grupo 1. Dessa forma, a distância da solução base em relação guia é maior, o que faz com que a Reconexão por Caminhos demande mais tempo para ser executada. Além disso, o tempo de execução para o Grupo 2 já é naturalmente mais elevado.

De acordo com os resultados, observou-se que o maior benefício em se utilizar a Reconexão por Caminhos foi justamente a de melhorar a qualidade da solução que é passada ao ILS-VNRD, facilitando a convergência na fase de busca local.

Testes computacionais também foram efetuados tendo como objetivo verificar a capacidade dos algoritmos em alcançar um dado valor alvo (ou seja, encontrar uma solução com um custo no mínimo tão bom quanto o valor alvo) em função do tempo. Os experimentos foram conduzidos conforme a proposta de [22]. Cada algoritmo foi executado 100 vezes, sendo interrompido após alcançar o valor alvo em cada execução ou esgotado um tempo limite, no caso, 800 segundos. Não foram permitidos tempos de execução repetidos; assim, os tempos repetidos foram descartados e uma nova execução foi agendada. Após as execuções, os tempos nos quais o valor foi atingido foram ordenados, de forma crescente, e para cada tempo t_i foi associada uma probabilidade $p_i = \frac{i-0,5}{100}$. A seguir, os pontos $z_i = (t_i, p_i)$, para $i = 1, \cdots, 100$ foram plotados. A Figura 1 mostra a distribuição de probabilidade empírica para os algoritmos AEH e ILS-MDR-IB, tendo como valor alvo a melhor solução de um problema-teste do grupo 2, no caso, gr229_VT45_T46_W138_25.

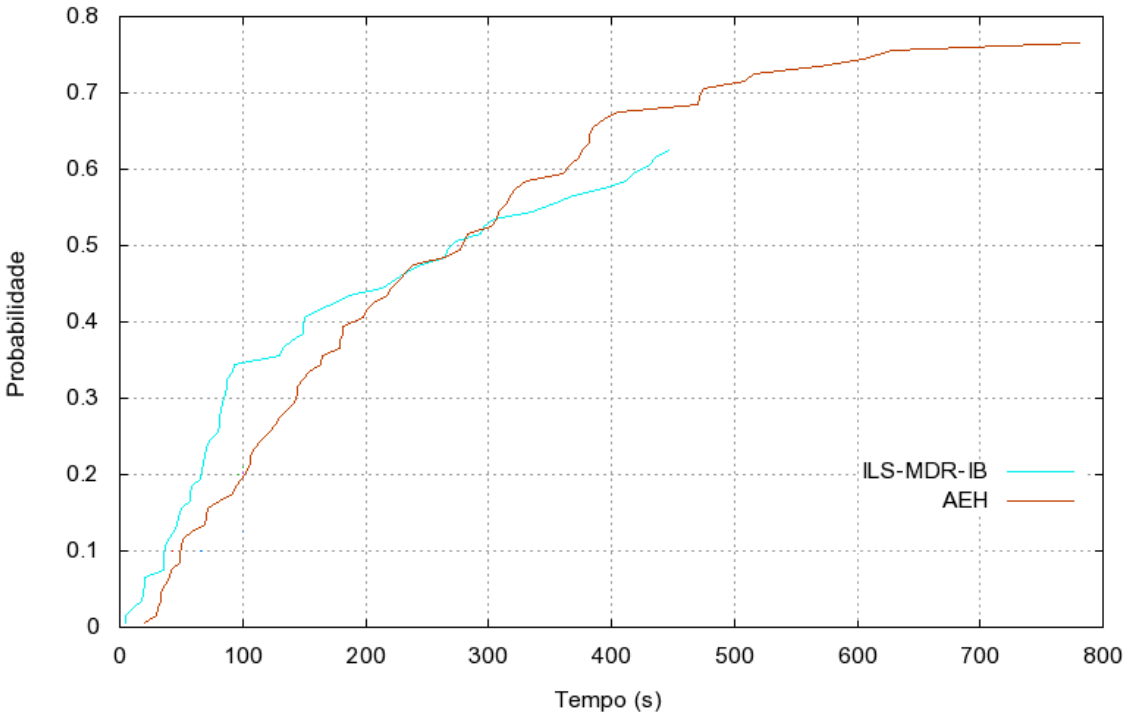

Figura 1: Distribuição de probabilidade empírica AEH $\times$ ILS-MDR

Verifica-se, pela Figura 1 que nenhum dos dois algoritmos alcança o alvo em 100% das execuções. No entanto, o Algoritmo Evolutivo Híbrido é o que o encontra com mais frequência, com quase 80% de probabilidade, contra pouco mais de 60% do algoritmo ILS-MDR-IB. Em relação ao tempo de execução, observa-se que até cerca de 250 segundos, o algoritmo ILS-MDR-IB tem uma probabilidade maior de alcançar o valor alvo; entretanto, depois desse tempo, o desempenho desse algoritmo passa a ser inferior ao do AEH. Após cerca de 450 segundos, o algoritmo ILS-MDR-IB estagna e não consegue mais alcançar o alvo, ao contrário do algoritmo AEH, o qual prossegue melhorando sua capacidade de encontrar o valor alvo.

De acordo com os resultados apresentados, o AEH foi o que gerou as melhores soluções para a maioria dos problemas-teste. Esse mérito se deve à característica do algoritmo em trabalhar com os dois tipos de busca, local e populacional, mesclando soluções de boa qualidade com soluções de qualidade inferior, fazendo com que a busca local tenha mais chances de melhorar a solução. O algoritmo se mostrou robusto por apresentar soluções finais com baixa variabilidade, no caso, sempre inferior a 0,14%.

5 CONCLUSÕES

Este trabalho abordou o Problema de Recobrimento de Rotas com Coleta de Prêmios (PRRCP). Para resolvê-lo, foi proposto um Algoritmo Evolutivo Híbrido, que mescla busca local com busca populacional. Para gerar os indivíduos da população foram adaptados cinco métodos de geração de solução inicial do PCV para serem utilizados no PRRCP. Para combinar diferentes indivíduos, foi utilizado o procedimento Reconexão por Caminhos e para refiná-los, utilizou-se a metaheurística *Iterated Local Search*, tendo o *Variable Neighborhood Random Descent* (VNRD) como método de busca local. O VNRD explora a vizinhança de uma solução por meio de dezessete diferentes estruturas de vizinhança, baseadas em movimentos clássicos usados para a resolução do PCV, bem como em adaptações da heurística GENIUS e do Método da Inserção Mais Barata. De acordo com os resultados obtidos, verifica-se que o algoritmo proposto é eficiente e robusto para resolver o PRRCP, sendo capaz de produzir soluções finais de boa qualidade.

Como trabalhos futuros, propõe-se um estudo aprofundado de técnicas exatas para resolução do PRRCP, como, por exemplo, geração de cortes e geração de colunas. A justificativa para tal é que estas técnicas já foram amplamente estudadas para o PCV e se mostraram bem sucedidas. Como o PRRCP é uma generalização do PCV, acredita-se que as mesmas sejam promissoras para resolução do PRRCP. Outra proposta consiste na otimização do desempenho da avaliação das soluções. Como a cada movimento realizado, toda a solução é reavaliada, seria apropriado o desenvolvimento de uma técnica que calculasse o novo custo tendo em vista apenas as modificações feitas na nova solução, o que contribuiria para reduzir o tempo demandado pelo algoritmo proposto.

AGRADECIMENTOS

Os autores agradecem à FAPERJ, FAPEMIG, CAPES e CNPq pelo apoio ao desenvolvimento do presente trabalho.

Referências

[1] H. R. Lourenço, O. C. Martin and T. Stützle. “Iterated Local Search”. In *Handbook of Metaheuristics*, edited by F. Glover and G. Kochenberger, chapter 11, pp. 321–353. Kluwer Academic Publishers, Boston, 2003.

[2] T. Stützle. “Iterated local search for the quadratic assignment problem”. *European Journal of Operational Research*, vol. 174, pp. 1519–1539, 2006.

[3] N. Mladenovic and P. Hansen. “Variable Neighboorhood Search”. *Computers and Operations Research*, vol. 24, pp. 1097–1100, 1997.

[4] P. Hansen and N. Mladenovic. “Variable neighborhood search: Principles and applications”. *European Journal of Operational Research*, vol. 130, pp. 449–467, 2001.

[5] P. Hansen, N. Mladenovic and J. A. M. Pérez. “Variable neighborhood search”. *European Journal of Operational Research*, vol. 191, pp. 593–595, 2008.

[6] M. Gendreau, A. Hertz and G. Laporte. “New Insertion and Post Optimization Procedures or The Traveling Salesman Problem”. *Operations Research*, vol. 40, pp. 1086–1094, 1992.

[7] I. C. M. Rosseti. “Estratégias sequenciais e paralelas de GRASP com reconexão por caminhos para o problema de síntese de redes a 2-caminhos”. Tese de doutorado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2003.

[8] J. R. Current. “Multiobjective Design of Transportation Networks”. Tese de doutorado, The Johns Hopkins University, Baltimore, USA, 1981.

[9] J. R. Current and D. A. Schilling. “The Covering Salesman Problem”. *Transportation Science*, vol. 23, pp. 208–213, 1989.

[10] J. R. Current and E. Rolland. “Efficient Algorithms for Solving the Shortest Covering Path Problem”. *Transportation Science*, vol. 28, pp. 317–327, 1994.

[11] M. Gendreau, G. Laporte and F. Semet. “The Covering Tour Problem”. *Operational Research*, vol. 45, pp. 568–576, 1995.

[12] E. Balas and A. Ho. “Set Covering Algorithms Using Cutting Planes, Heuristics, and Subgradient Optimizations: A Computational Study”. *Mathematical Programming*, vol. 12, pp. 37–70, 1980.

[13] R. Baldacci, M. A. Boschetti, V. Maniezzo and M. Zamboni. “Scatter search methods for the covering tour problem”. In *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*, edited by C. Rego and B. Alidaee, volume 30 of *Operations Research/Computer Science Interfaces Series*, pp. 59–91. Kluwer Academic Publishers, Boston, 2005.

[14] L. C. S. Motta. “Novas abordagens para o Problema de Recobrimento de Rotas”. Dissertação de mestrado, Programa de Pós-Graduação em Computação, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, 2001. Disponível em http://www.ic.uff.br/PosGraduacao/Dissertacoes/187.pdf.

[15] L. C. Motta, L. T. Nogueira and L. S. Ochi. "Improving performance of algorithms for the covering tour problem by applying reduction rules". In *Proceedings of the 25th Mini EURO Conference on Uncertainty and Robustness in Planning and Decision Making – URPDM 2010*, University of Coimbra, Portugal, 2010.

[16] A. R. Lyra. "O Problema de Recobrimento de Rotas com Coleta de Prêmios: Regras de Redução, Formulação Matemática e Heurísticas". Dissertação de mestrado, Programa de Pós-Graduação em Computação, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, 2004. Disponível em http://www.ic.uff.br/PosGraduacao/Dissertacoes/112.pdf.

[17] T. Feo and M. Resende. "Greedy Randomized Search Procedure". *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.

[18] M. S. A. Silva. "Problema de Recobrimento de Rotas com Coleta de Prêmios". Dissertação de mestrado, Programa de Pós-Graduação em Computação, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, 2009. Disponível em http://www.ic.uff.br/PosGraduacao/Dissertacoes/417.pdf.

[19] A. E. Eiben and G. Rudolph. "Theory of Evolutionary Algorithms: A Birds Eye View". *Theoretical Computer Science*, vol. 229, pp. 3–9, 1999.

[20] A. Hertz and D. Kobler. "A framework for the description of evolutionary algoritms". *European Journal of Operations Research*, vol. 126, pp. 1–12, 2000.

[21] G. Reinelt. "TSPLIB - A Traveling Salesman Problem Library". *ORSA - Journal on Computing*, pp. 376–384, 1991.

[22] R. M. Aiex, M. G. C. Resende and C. C. Ribeiro. "Probability distribution of solution time in GRASP: an experimental investigation". *Journal of Heuristics*, vol. 8, pp. 343–373, 2002.