UMA HEURÍSTICA BASEADA EM GRASP E *ITERATED LOCAL SEARCH* PARA O PROBLEMA DA MÍNIMA LATÊNCIA

Marcos de Melo da Silva[1], **Anand Subramanian**[1,2], **Luiz Satoru Ochi**[1]
[1]Instituto de Computação - Universidade Federal Fluminense
Rua Passo da Pátria, 156, Bloco E, 3º andar, São Domingos, 24210-240, Niterói, RJ
{mmsilva, anand, satoru}@ic.uff.br
[2]Departamento de Engenharia de Produção - Universidade Federal da Paraíba
Centro de Tecnologia, Campus I - Bloco G, Cidade Universitária, 58051-970, João Pessoa, PB
anand@ct.ufpb.br

RESUMO

O Problema da Mínima Latência (PML) é uma generalização do Problema do Caixeiro Viajante (PCV) onde a ordem em que os clientes são visitados afeta diretamente no custo final da solução. O objetivo é minimizar a latência total de todos os clientes, onde a latência de um cliente é dada pelo tempo necessário para percorrer o caminho com início no depósito e término em tal cliente. Neste trabalho é proposto um algoritmo baseado nas metaheurísticas *Greedy Randomized Adaptive Search Procedure* (GRASP) e *Iterated Local Search* (ILS), que utiliza o método *Variable Neighborhood Descent* (VND) com ordem aleatória de vizinhanças (RVND) na fase de busca local. A heurística desenvolvida foi testada em dois conjuntos de instâncias disponíveis na literatura. Os resultados obtidos pelo algoritmo, em termos de qualidade das soluções, foram competitivos, sendo capaz de alcançar ou superar as melhores soluções conhecidas.

PALAVRAS CHAVE. Heurística híbrida, Mínima Latência, GRASP, *Iterated Local Search*.

ABSTRACT

The Minimum Latency Problem (MLP) is a generalization of the Traveling Salesman Problem (TSP) where the order in which the customers are visited directly affects the solution cost. The objective is to minimize the customers total latency, where the latency of a customer is given by the time necessary to traverse the path that begins at the depot and ends at the corresponding customer. This work proposes an algorithm based on the metaheuristics Greedy Randomized Adaptive Search Procedure (GRASP) and Iterated Local Search (ILS) that uses the Variable Neighborhood Descent with Random Neighborhood Ordering (RVND) method in the local search phase. The developed heuristic was tested in two set of instances available in the literature. The results obtained by the algorithm, in terms of solution quality, were competitive, being capable to equal or to improve the best known solutions.

KEY WORDS. Hybrid Heuristic, Minimum Latency, GRASP, Iterated Local Search.

1 Introdução

O Problema da Mínima Latência (PML) é uma generalização do Problema do Caixeiro Viajante (PCV) e pode ser definido da seguinte forma. Seja $G = (V \cup \{0\}, A)$ um grafo completo orientado, onde $V = \{1, \ldots, n\}$ é o conjunto de vértices e $A = \{(i, j) \mid i, j \in V \cup \{0\}, i \neq j\}$ é o conjunto de arcos com custo associado $c_{ij},\ (i, j) \in A$. O vértice 0 é a origem ou depósito. Define-se a latência de um vértice $i \in V$ como sendo o tempo necessário para percorrer o caminho com origem no depósito e término em tal vértice. O PML busca um circuito hamiltoniano cuja latência total seja mínima, sendo esta determinada pela soma da latência de todos os vértices. Assume-se que a latência do depósito é zero e o tempo de visita está incluído no custo c_{ij}.

O PML foi provado ser NP-Difícil para espaços métricos gerais (Sahni e Gonzalez, 1976) e também para o caso em que a estrutura subjacente é uma árvore com pesos nas arestas (Sitters, 2002). Para estruturas como caminhos, árvores não ponderadas, árvores de diâmetro 3, diversos autores apresentam algoritmos de tempo polinômial utilizando principalmente programação dinâmica (Blum *et al*., 1994; García *et al*., 2002; Wu *et al*., 2004). Alguns dos sinônimos adotados na literatura para o PML são *Traveling Repairman Problem* (Tsitsiklis, 1992), *Delivery man Problem* (Fischetti *et al*., 1993), Problema do caixeiro viajante com custos cumulativos (Bianco *et al*., 1993), *School Bus Driver Problem* (Chaudhuri *et al*., 2003). Na literatura são encontradas duas variações do PML com relação ao tipo de percurso utilizado. A primeira versão é dada por um caminho hamiltoniano com início no vértice 0. A segunda versão considera um circuito hamiltoniano com início e fim no depósito.

Apesar do PML parecer uma simples variante do PCV verifica-se que o primeiro possui propriedades não presentes no segundo. Uma delas é que pequenas alterações locais na configuração dos pontos de entrada pode levar a mudanças não locais significativas na estrutura da solução ótima (Blum *et al*., 1994; Goemans e Kleinberg, 1998). Outra característica do PML é o carater não local da função objetivo. Um arco adicional inserido no início do circuito afeta a latência de todos os vértices restantes (Arora e Karakostas, 2003). O PML também considera tempos de espera (latência) de um sistema de serviço do ponto de vista do cliente, ou seja, enquanto no PML o objetivo é minimizar o tempo médio que cada cliente teria que esperar até ser atendido, no PCV este objetivo é minimizar o tempo total gasto para visitar todos os clientes. Isto posto, diz-se que o PML é orientado ao cliente, enquanto o PCV é orientado ao servidor (Archer e Levin, 2003). Desta forma, o PML pode ser empregado na modelagem de vários tipos de sistemas de serviço. Tsitsiklis (1992) observa que a função de custo utilizada no PML é a mesma que a *flowtime*, também conhecida como soma dos tempos de conclusão, sendo esta uma medida de desempenho muito utilizada na teoria de *scheduling*.

O PML e o PCV são casos especiais do problema mais geral conhecido como *Time-Dependent Traveling Salesman Problem* (TDTSP). Neste último, o custo associado ao percurso entre dois vértices depende da localização destes no espaço métrico bem como da posição em que eles aparecem no circuito. O objetivo é minimizar o custo total de visitar todos os nós (Abeledo *et al*., 2010b,a; Blum *et al*., 1994; Lucena, 1990). O TDTSP e o PML podem ser vistos também como o problema de escalonamento de uma máquina com tempos de processamento dependentes da sequência (Bigras *et al*., 2008; Gouveia e Voss, 1995; Picard e Queyranne, 1978). Na literatura são tratados também algumas variações do PML. A versão com janelas de tempo é estudada por Heilporn *et al*. (2010), Tsitsiklis (1992) e Van Eijl (1995). O caso onde os custos são assimétricos é estudado por Nagarajan e Ravi (2008). A versão com multiplos servidores é tratada por Fakcharoenphol *et al*. (2007) e a versão ponderada por García *et al*. (2002) e Wu (2000). Ausiello *et al*. (2000) associa o PML com o problema de busca em grafo (*Graph Searching Problem*).

Aplicações do PML podem ser encontradas com frequência em situações de distribuição onde algum tipo de critério de qualidade no atendimento aos clientes deve ser enfocado. Um

exemplo prático é a entrega de pizzas, onde vários pedidos são agrupados e deseja-se minimizar o tempo de chegada até os clientes (Méndez-Díaz *et al*., 2008). Outras aplicações podem ser encontradas na área de redes de computadores. Por exemplo, ao se fazer uma busca por uma determinada informação armazenada em algum lugar da rede (Ezzine *et al*., 2010).

Alguns algoritmos exatos para o PML foram propostos na literatura. Lucena (1990) propõe um algoritmo enumerativo baseado em uma formulação inteira não-linear na qual os limites inferiores são derivados a partir de uma relaxação lagrangeana. Bianco *et al*. (1993) desenvolvem dois algoritmos exatos que incorporam limites inferiores obtidos a partir da relaxação lagrangeana de uma formulação de programação inteira. Fischetti *et al*. (1993) desenvolvem um algoritmo enumerativo no qual são embutidos limites inferiores obtidos a partir de formulações de programação linear inteira. Van Eijl (1995) apresenta uma formulação em programação inteira mista e também uma adaptação desta para a variante com janelas de tempo. Wu *et al*. (2004) apresentam algoritmos exatos que combinam programação dinâmica e *Branch and Bound*. Méndez-Díaz *et al*. (2008) apresentam uma nova formulação de programação linear inteira, além de sugerirem um conjunto de desigualdades válidas utilizadas em um algoritmo de plano de cortes. Ezzine *et al*. (2010) apresentam duas novas formulações de programação linear inteira e testam a qualidade das relaxações lineares. Bigras *et al*. (2008) apresentam várias formulações de programação inteira bem como um algoritmo de *Branch-and-Bound*. Abeledo *et al*. (2010a,b) desenvolvem um algoritmo de *Branch-Cut-and-Price* a partir de uma formulação estendida bem como várias famílias de desigualdades definidoras de facetas. As maiores instâncias resolvidas de forma exata até agora tem até 107 clientes e foram selecionadas da TSPLIB (Reinelt, 1991) por Abeledo *et al*. (2010a,b).

Diversos autores propuseram algoritmos aproximativos para o PML (Archer e Blasiak, 2010; Archer e Levin, 2003; Arora e Karakostas, 2003; Ausiello *et al*., 2000; Blum *et al*., 1994; Chaudhuri *et al*., 2003; Fakcharoenphol *et al*., 2007; Goemans e Kleinberg, 1998; Nagarajan e Ravi, 2008). O primeiro foi desenvolvido por Blum *et al*. (1994) com um fator de aproximação de 144. Para espaços métricos gerais, o menor fator de aproximação conseguido até o momento é de 3,59 cujo algoritmo foi desenvolvido por Chaudhuri *et al*. (2003). Para o caso onde uma árvore ponderada por arestas é considerado, o menor fator de aproximação é de 3,03 e foi obtido por Archer e Blasiak (2010).

Recentemente, Salehipour *et al*. (2011) propuseram um algoritmo heurístico para o PML baseado na metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP) (Feo e Resende, 1995), *Variable Neighborhood Descent* (VND) e *Variable Neighborhood Search* (VNS) (Mladenović e Hansen, 1997). Os autores testaram o algoritmo em conjuntos de instâncias gerados aleatoriamente contendo até 1000 clientes.

Este trabalho propõe um algoritmo baseado nas metaheurísticas GRASP e *Iterated Local Search* (ILS) (Lourenço *et al*., 2003), utilizando como método de busca local o VND com ordem aleatória de vizinhanças (RVND) (Subramanian *et al*., 2010) para tratar o problema com as duas variações citadas.

O restante do trabalho está organizado da seguinte forma. A Seção 2 apresenta uma formulação matemática para PML. A Seção 3 descreve o algoritmo proposto. Os resultados estão contidos na Seção 4. As considerações finais são feitas na Seção 5.

2 Formulação Matemática

A formulação matemática para o PML proposta por Van Eijl (1995) é apresentada a seguir. Dado o grafo descrito anteriomente, define-se a variável binária x_{ij} que indica se o arco $(i, j) \in A$ está incluído ou não no circuito. A variável t_{ij} assume o valor do tempo de partida do nó $i \in V$ se $x_{ij} = 1$ ou 0, caso contrário. M denota uma constante de valor suficientemente grande.

$$\text{Minimizar} \sum_{i=1}^{n} \sum_{j=0, j\neq i}^{n} t_{ij} \tag{1}$$

Sujeito a:

$$\sum_{j=0, j\neq i}^{n} x_{ij} = 1, \qquad \forall i = 0, \ldots, n \tag{2}$$

$$\sum_{i=0, i\neq j}^{n} x_{ij} = 1, \qquad \forall j = 0, \ldots, n \tag{3}$$

$$\sum_{i=1, i\neq j}^{n} t_{ij} + \sum_{i=0, i\neq j}^{n} c_{ij} x_{ij} = \sum_{k=0, k\neq j}^{n} t_{jk} \qquad \forall j = 1, \ldots, n \tag{4}$$

$$0 \leq t_{ij} \leq M x_{ij} \qquad \forall i, j = 0, \ldots, n,\ i \neq j,\ i \neq 0 \tag{5}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j = 0, \ldots, n,\ i \neq j \tag{6}$$

A função objetivo (1) minimiza a soma dos tempos de partida. As restrições (2) e (3) asseguram que cada vértice, incluindo o depósito, é visitado exatamente uma vez. As restrições (4) indicam que, se $x_{ij} = 1$, então o tempo de partida do vértice j é igual ao tempo de partida do vértice i mais o tempo de viagem c_{ij}. Verifica-se também que tais restrições irão evitar a ocorrência de sub-rotas. Se M é um limite superior sobre o tempo de partida do vértice i então (5) é válida quando $x_{ij} = 1$. Além disso, tais restrições forçam que $t_{ij} = 0$ se $x_{ij} = 0$. As restrições (5) e (6) definem o domínio das variáveis.

3 Algoritmo proposto

O algoritmo proposto para o PML, denominado GILS, é detalhado nesta seção. Este reúne componentes das metaheurísticas GRASP e ILS, empregando o procedimento VND com ordem aleatória na escolha das vizinhanças (RVND) na fase de busca local. Conforme pode ser observado no Algoritmo 1, a heurística *multi-start* executa $maxIter$ iterações (linhas 3 a 20), onde em cada uma delas uma solução inicial é gerada por um método que tem como base a fase de construção do GRASP (linha 4). Em seguida, o laço principal do ILS (linhas 7 a 15) tenta melhorar a solução gerada utilizando na etapa de busca local um procedimento RVND (linha 8) combinado com um mecanismo de pertubação (linha 13). Quando uma solução de melhor qualidade é encontrada o contador de iterações do ILS (*iterILS*) é reiniciado (linhas 9 a 12). O número máximo de pertubações sem melhora é dada pelo parâmetro $maxIterILS$. Observa-se também que a pertubação é realizada sempre sobre a melhor solução corrente s' de uma dada iteração. O algoritmo retorna a melhor solução s^* encontrada entre todas as iterações.

3.1 Construção de soluções iniciais

Uma solução inicial viável para o PML é gerada por um procedimento construtivo que utiliza uma abordagem gulosa randomizada baseada na fase de construção do GRASP. O pseudocódigo do procedimento desenvolvido está descrito no Algoritmo 2. Primeiramente uma solução parcial s é inicializada com o vértice referente ao depósito (linha 2) e a Lista de Candidadatos (LC) inicializada com os vértices ainda não pertencentes a s (linhas 3 e 4). No laço principal (linhas 6 a 13), os vértices em LC são ordenados de acordo com o critério de vizinho mais próximo (menor custo) tendo como referência o último cliente adicionado a s (linha 7), de

Algoritmo 1: GILS

```
1  Procedimento GILS(maxIter, maxIterILS, α)
2  f* ← ∞;
3  para i ← 1, ..., maxIter faça
4      s ← Construcao(α);
5      s' ← s;
6      iterILS ← 0;
7      enquanto iterILS < maxIterILS faça
8          s ← RVND(s); /* nv = #vizinhanças */
9          se f(s) < f(s') então
10             s' ← s;
11             iterILS ← 0;
12         fim se
13         s ← Pertuba(s');
14         iterILS ← iterILS + 1;
15     fim enqto
16     se f(s') < f* então
17         s* ← s';
18         f* ← f(s');
19     fim se
20 fim para
21 retorna s*;
22 fim GILS
```

forma que no passo seguinte somente os $\alpha\%$ candidatos mais próximos serão colocados na Lista Restrita de Candidatos (LRC) (linha 8). Em seguida, um vértice é selecionado aleatoriamente de LRC e adicionado a solução s (linhas 9 e 10). O laço principal termina quando todos os clientes em LC tenham sido inseridos em s.

Algoritmo 2: Construcao

```
1  Procedimento Construcao(α)
2  s ∪ {0};
3  Inicializar lista de candidatos LC;
4  LC ← LC − {0};
5  r ← 0;
6  enquanto LC ≠ ∅ faça
7      Ordene LC em ordem crescente de custo com relação a r;
8      Atualize LRC com os α% primeiros clientes de LC;
9      Selecione aleatoriamente c ∈ LRC;
10     s ∪ {c};
11     r ← c;
12     LC ← LC − {r};
13 fim enqto
14 retorna s;
15 fim Construcao
```

3.2 Busca local

A fase de busca local é realizada por um método baseado no procedimento VND com escolha aleatória da ordem em que as estruturas de vizinhanças serão aplicadas (RVND). Seja

t o número de estruturas vizinhanças e $N = \{N^1, N^2, N^3, \ldots, N^t\}$, o RVND selecionará aleatoriamente a ordem em que estas estruturas serão executadas. Testes preliminares mostraram que esta abordagem, em média, encontra resultados de melhor qualidade quando comparado com versões que utilizam ordem determinística.

O algoritmo proposto possui um conjunto de cinco estruturas de vizinhanças amplamente exploradas na literatura, a saber:

- ***Swap*** $-$ $N^{(1)}$ $-$ Permutação entre dois clientes.
- **2-*opt*** $-$ $N^{(2)}$ $-$ Dois arcos não adjacentes são removidos e outros dois são adicionados formando um novo percurso.
- **Reinserção** $-$ $N^{(3)}$ $-$ Um único cliente é removido e inserido em outra posição do percurso.
- **Or-*opt2*** $-$ $N^{(4)}$ $-$ Dois clientes adjacentes são removidos e inseridos em outra posição do percurso.
- **Or-*opt3*** $-$ $N^{(5)}$ $-$ Três clientes adjacentes são removidos e inseridos em outra posição do percurso.

Cada uma destas estruturas é examinada exaustivamente e somente o movimento de melhora mais significante para cada vizinhança é considerado.

O pseudocódigo do procedimento RVND é apresentado no Algoritmo 3. Inicialmente, a Lista de Vizinhanças (LV) é inicializada com as estruturas de vizinhança (linha 2). A cada execução do laço principal do método (linhas 3 a 13), uma estrutura de vizinhança N^{η} é selecionada aleatoriamente de LV (linha 4) e então o melhor vizinho encontrado ao aplicar esta vizinhança é armazenado em s' (linha 5). Em caso de melhora, LV é reinicializada com todas as estruturas de vizinhança (linhas 6 a 10). Caso contrário, N^{η} é removida de LV (linha 11). O procedimento termina quando LV estiver vazia.

Algoritmo 3: RVND

```
1  Procedimento RVND(s)
2  Inicializar lista de vizinhança LV;
3  enquanto LV ≠ ∅ faça
4      Selecione aleatoriamente a vizinhança N^(η) ∈ LV;
5      Encontre o melhor vizinho s' de s ∈ N^(η);
6      se f(s') < f(s) então
7          s ← s';
8          f(s) ← f(s');
9          Atualiza LV;
10     senão
11         Remova N^(η) de LV;
12     fim se
13 fim enqto
14 retorna s;
15 fim RVND
```

3.3 Mecanismo de perturbação

O mecanismo de perturbação utilizado é descrito no Algoritmo 4. Este é baseado no procedimento *double-bridge* que foi originalmente desenvolvido para o PCV e consiste na remoção

de quatro arcos de um determinado percurso e na inserção de outros quatro de forma a gerar um novo percurso (Martin *et al.*, 1991). Este mecanismo pode ser visto também como uma permutação entre dois segmentos disjuntos de um percurso. A cada execução do procedimento dois segmentos disjuntos A e B de s' são selecionados aleatoriamente (linha 3) e permutados (linha 4).

Algoritmo 4: Perturba

```
1 Procedimento Perturba(s)
2 s' ← s;
3 Selecione aleatoriamente dois segmentos de percurso disjuntos A e B de s';
4 s ← Permute os segmentos A e B de s';
5 retorna s;
6 fim Perturba
```

4 Resultados computacionais

O algoritmo GILS foi implementado na linguagem C++ (g++ 4.4.3) e executado em um Intel® Core™ i7 com 2.93 GHz, 8.0 GB de memória RAM e sistema operacional GNU/Linux Ubuntu 10.04 (*kernel* 2.6.32-25). O procedimento foi testado em dois conjuntos de instâncias selecionadas da TSPLIB (Reinelt, 1991) por Abeledo *et al*. (2010a,b) e Salehipour *et al*. (2011). O primeiro conjunto é composto por 22 problemas-teste variando entre 42 e 107 clientes. O segundo por 10 problemas-teste variando entre 70 e 532 clientes. Para as duas variações do PML com relação ao tipo de percurso utilizado, Abeledo *et al*. (2010a,b) tratam a versão que considera um circuito hamiltoniano com início e fim no depósito e Salehipour *et al*. (2011) a versão dada por um caminho hamiltoniano com início no vértice 0.

O número de iterações ($maxIter$) foi setado em 10 e o número de pertubações ($maxIterILS$) em n, sendo este igual ou ao número de clientes ou ao tempo em segundos (o que ocorrer primeiro). Para o parâmetro α utilizado na fase de construção foi utilizado o valor 20%. Estes valores foram calibrados empiricamente por meio de testes preliminares. Para cada instância foram realizadas 25 execuções do GILS.

Nas tabelas apresentadas a seguir, **Problema** indica o nome da instância, **Best Sol.** a melhor solução encontrada pelo respectivo trabalho, **Avg. Gap** o *gap* entre a solução média obtida pelo GILS e a melhor solução da literatura, **Avg. Sol.** a média das soluções obtidas, **UB** o limite superior obtido pelo algoritmo exato proposto por Abeledo *et al*. (2010a,b), **Avg. Time** a média dos tempos, em segundos, das 25 execuções. As Tabelas 1 e 2 mostram, respectivamente, os resultados computacionais para as instâncias selecionadas por Abeledo *et al*. (2010a,b) e por Salehipour *et al*. (2011).

A Tabela 1 ilustra os resultados obtidos pelo GILS para as instâncias selecionadas por Abeledo *et al*. (2010a,b). Uma comparação é feita com o algoritmo exato proposto pelos mesmos autores. Os resultados mostram que o algoritmo proposto foi capaz de alcançar todas as soluções ótimas e para as duas instâncias que o algoritmo exato não foi capaz de encontrar o ótimo, o GILS obteve soluções de melhor qualidade. Ao comparar os resultados obtidos, observa-se que o desempenho do GILS foi bastante satisfatório, principalmente na qualidade das soluções médias.

A Tabela 2 apresenta os resultados obtidos pelo GILS para as instâncias selecionadas por Salehipour *et al*. (2011). Uma comparação é feita com o algoritmo heurístico proposto por estes autores. Os resultados mostram que o GILS foi capaz de alcançar as mesmas soluções ou melhorá-las. Ao comparar os resultados obtidos com os da literatura, observa-se que o desempenho do GILS foi bastante satisfatório. Não foi possível fazer uma comparação direta dos tempos

computacionais apresentados pelo algoritmo devido aos diferentes ambientes computacionais e abordagens empregados.

Tabela 1: Resultados computacionais para as instâncias de Abeledo *et al*. (2010a,b)

	Abeledo *et. al.*	GILS			
Problema	UB	Best Sol.	Avg. Sol.	Avg. Gap	Avg. Time
dantzig42	**12528**	**12528**	12528,00	0,00	2,04
swiss42	**22327**	**22327**	22327,00	0,00	2,01
att48	**209320**	**209320**	209320,00	0,00	4,48
gr48	**102378**	**102378**	102378,00	0,00	3,61
hk48	**247926**	**247926**	247926,00	0,00	4,34
eil51	**10178**	**10178**	10178,00	0,00	5,79
berlin52	**143721**	**143721**	143721,00	0,00	5,45
brazil58	**512361**	**512361**	512361,00	0,00	10,57
st70	**20557**	**20557**	20557,00	0,00	21,64
eil76	**17976**	**17976**	17976,00	0,00	40,37
pr76	**3455242**	**3455242**	3455242,00	0,00	25,19
gr96	**2097170**	**2097170**	2097170,00	0,00	77,72
rat99	58288	**57986**	57986,00	-0,51	121,92
kroA100	**983128**	**983128**	983128,00	0,00	115,74
kroB100	**986008**	**986008**	986008,00	0,00	117,46
kroC100	**961324**	**961324**	961324,00	0,00	89,01
kroD100	**976965**	**976965**	976965,00	0,00	106,47
kroE100	**971266**	**971266**	971266,00	0,00	91,81
rd100	**340047**	**340047**	340047,00	0,00	101,16
eil101	27519	**27513**	27513,00	-0,02	137,71
lin105	**603910**	**603910**	603910,00	0,00	106,52
pr107	**2026626**	**2026626**	2026626,00	0,00	119,47

Tabela 2: Resultados computacionais para as instâncias de Salehipour *et al*. (2011)

	Salehipour *et. al.*		GILS			
Problema	Best Sol.	Time*	Best Sol.	Avg. Sol.	Avg. Gap	Avg. Time
st70	19553	2,14	**19215**	19215,00	-1,72	17,29
rat99	56994	13,98	**54984**	54984,00	-3,52	139,05
kroD100	976830	4,69	**949594**	949594,00	-2,78	84,96
lin105	**585823**	11,25	**585823**	585823,00	0,00	90,54
pr107	1983475	15,28	**1980767**	1980767,00	-0,13	138,92
rat195	213371	104,38	**210191**	210317,44	-1,43	1940,95
pr226	7226554	228,78	**7100308**	7100308,00	-1,74	2264,06
lin318	5876537	401,06	**5560679**	5579800,88	-5,04	3408,92
pr439	18567170	508,84	**17812403**	17921625,16	-3,47	5200,11
att532	18448435	5134,36	**5694544**	5723481,28	-68,97	7134,49

* Pentium 2.4 GHz

5 Considerações finais

Este trabalho propôs uma heurística híbrida baseada em GRASP, ILS e RVND para o PML. Observou-se que a abordagem desenvolvida é simples e sua eficácia, em termos de qualidade das soluções, foi demonstrado por meio de experimentos realizados em dois conjuntos de instâncias compostos por 32 problemas-teste com até 532 clientes. O método apresentado foi capaz de melhorar o resultado de 11 soluções e de chegar à mesma solução conhecida nas outras 21 instâncias.

Como trabalhos futuros, pretende-se extender o algoritmo proposto para tratar outras variantes do PML tal como o PML com coleta de prêmios e a versão com multiplos veículos.

Referências

Abeledo, H., Fukasawa, R., Pessoa, A. e Uchoa, E. The time dependent traveling salesman problem: Polyhedra and algorithm. Relatório técnico. URL `http://www.optimization-online.org/DB_HTML/2010/12/2872.html`, 2010a.

Abeledo, H., Fukasawa, R., Pessoa, A. e Uchoa, E. The time dependent traveling salesman problem: Polyhedra and branch-cut-and-price algorithm. Festa, P. (Ed.), *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, p. 202–213. Springer Berlin / Heidelberg, 2010b.

Archer, A. e Blasiak, A. Improved approximation algorithms for the minimum latency problem via prize-collecting strolls. *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, p. 429–447, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics, 2010.

Archer, A. e Levin, A. Faster approximation algorithms for the minimum latency problem. *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, p. 88–96. SIAM, 2003.

Arora, S. e Karakostas, G. (2003), Approximation schemes for minimum latency problems. *SIAM Journal on Computing*, v. 32, p. 1317–1337.

Ausiello, G., Leonardi, S. e Marchetti-Spaccamela, A. On salesmen, repairmen, spiders, and other traveling agents. *Proceedings of the 4th Italian Conference on Algorithms and Complexity*, CIAC '00, p. 1–16, London, UK. Springer-Verlag, 2000.

Bianco, L., Mingozzi, A. e Ricciardelli, S. (1993), The traveling salesman problem with cumulative costs. *Networks*, v. 23, n. 2, p. 81–91.

Bigras, L.-P., Gamache, M. e Savard, G. (2008), The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization*, v. 5, p. 685–699.

Blum, A., Chalasanit, P., Pulleyblankt, B., Raghavan, P. e Sudan, M. The minimum latency problem. *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, STOC '94, p. 163–171, New York, NY, USA. ACM, 1994.

Chaudhuri, K., Godfrey, B., Rao, S. e Talwar, K. Paths, trees, and minimum latency tours. *44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS'03, p. 36–45, 2003.

Ezzine, I. O., Semet, F. e Chabchoub, H. New formulations for the traveling repairman problem. *8th International Conference of Modeling and Simulation*, MOSIM' 10, 2010.

Fakcharoenphol, J., Harrelson, C. e Rao, S. (2007), The k-traveling repairmen problem. *ACM Transactions on Algorithms*, v. 3.

Feo, T. A. e Resende, M. G. C. (1995), Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109–133.

Fischetti, M., Laporte, G. e Martello, S. (1993), The delivery man problem and cumulative matroids. *Operations Research*, v. 41, n. 6, p. 1055–1064.

García, A., Jodrá, P. e Tejel, J. (2002), A note on the traveling repairman problem. *Networks*, v. 40, n. 1, p. 27–31.

Goemans, M. X. e Kleinberg, J. M. (1998), An improved approximation ratio for the minimum latency problem. *Mathematical Programming*, v. 82, p. 111–124.

Gouveia, L. e Voss, S. (1995), A classification of formulations for the (time-dependent) traveling salesman problem. *European Journal of Operational Research*, v. 83, n. 1, p. 69–82.

Heilporn, G., Cordeau, J.-F. e Laporte, G. (2010), The delivery man problem with time windows. *Discrete Optimization*, v. 7, p. 269–282.

Lourenço, H., Martin, O. e Stützle, T. Iterated local search. Glover, F. e Kochenberger, G. (Eds.), *Handbook of Metaheuristics*, p. 320–353. Kluwer Academic Publishers, 2003.

Lucena, A. (1990), Time-dependent traveling salesman problem - the deliveryman case. *Networks*, v. 20, n. 6, p. 753–763.

Martin, O., Otto, S. W. e Felten, E. W. (1991), Large-step markov chains for the traveling salesman problem. *Complex Systems*, v. 5, p. 299–326.

Méndez-Díaz, I., Zabala, P. e Lucena, A. (2008), A new formulation for the traveling deliveryman problem. *Discrete Applied Mathematics*, v. 156, n. 17, p. 3223–3237.

Mladenović, N. e Hansen, P. (1997), Variable neighborhood search. *Computers and Operations Research*, v. 24, p. 1097–1100.

Nagarajan, V. e Ravi, R. The Directed Minimum Latency Problem. Goel, A., Jansen, K., Rolim, J. D. P. e Rubinfeld, R. (Eds.), *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques.*, volume 5171 of *Lecture Notes in Computer Science*, p. 193–206. Springer, 2008.

Picard, J.-C. e Queyranne, M. (1978), The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research*, v. 26, n. 1, p. 86–110.

Reinelt, G. (1991), Tsplib–a traveling salesman problem library. *Informs Journal on Computing*, v. 3, n. 4, p. 376–384.

Sahni, S. e Gonzalez, T. (1976), P-complete approximation problems. *Journal of The ACM*, v. 23, n. 3, p. 555–565.

Salehipour, A., Sörensen, K., Goos, P. e Bräysy, O. (2011), Efficient grasp+vnd and grasp+vns metaheuristics for the traveling repairman problem. *4OR: A Quarterly Journal of Operations Research*, v. , p. 1–21.

Sitters, R. The minimum latency problem is np-hard for weighted trees. *Proceedings of the 9th International IPCO Conference on Integer Programming and Combinatorial Optimization*, p. 230–239, London, UK. Springer-Verlag, 2002.

Subramanian, A., Drummond, L. M. A., Bentes, C., Ochi, L. S. e Farias, R. (2010), A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers and Operations Research*, v. 37, p. 1899–1911.

Tsitsiklis, J. N. (1992), Special cases of traveling salesman and repairman problems with time windows. *Networks*, v. 22, n. 3, p. 263–282.

Van Eijl, C. A. A polyhedral approach to the delivery man problem. Relatório Técnico COSOR 95-19, Eindhoven University of Technology, 1995.

Wu, B. Y. (2000), Polynomial time algorithms for some minimum latency problems. *Information Processing Letters*, v. 75, n. 5, p. 225–229.

Wu, B. Y., Huang, Z.-N. e Zhan, F.-J. (2004), Exact algorithms for the minimum latency problem. *Information Processing Letters*, v. 92, n. 6, p. 303–309.