UM ALGORITMO MEMÉTICO PARA A SOLUÇÃO DO PROBLEMA DE MÍNIMA LATÊNCIA

Luis Eduardo Amorim, Luciana Brugiolo Gonçalves, Salles Viana Gomes de Magalhães
Departamento de Informática - Universidade Federal de Viçosa
Centro de Ciências Exatas e Tecnológicas, Campus da UFV, 36570-000, Viçosa, MG
{luis.amorim,lbrugiolo,salles}@ufv.br

RESUMO

Este trabalho propõe um algoritmo genético em conjunto com um modelo de busca local para a solução do problema Problema da Mínima Latência (PML). O objetivo do problema é encontrar uma ordem de visitação cuja soma dos tempos de espera de um conjunto de clientes seja minimizada. Para a busca local, foram utilizados duas estruturas de vizinhança amplamente utilizada em problemas de roteamento: 2-opt e reinserção. A heurística proposta foi testada em dois conjuntos de instâncias da literatura e apresentou resultados preliminares competitivos tanto em termos de qualidade da solução quanto em termos do tempo de processamento, sendo em alguns casos até 5 vezes mais rápido do que o melhor método descrito em literatura.
PALAVRAS CHAVE. Otimização Combinatória, Algoritmo Genético, Problema Mínima Latência.

Áreas Principais: Metaheurísticas, Otimização Combinatória, Logística e Transportes.

ABSTRACT

This work presents a genetic algorithm with local search for solving the Minimum Latency Problem (*MLP*). The *MLP*'s goal is to find a visitation order to customers where the total waiting time is minimized. The local search was designed with two neighborhood structures commonly used in routing problems: 2-opt and insertion. The proposed heuristic was tested using two instance sets from literature and it was able to find high quality solutions in a competitive amount of time, being up to 5 times faster than the best known method from literature.
KEY WORDS: Combinatorial Optimization, Genetic Algorithm, Minimum Latency Problem.

Main areas: Metaheuristics, Combinatorial Optimization, Logistics and Transport.

1 Introdução

O Problema da Mínima Latência (PML) pode ser descrito como uma variante do Problema do Caixeiro Viajante (PCV) onde o foco está relacionado ao cliente. Enquanto no PCV o objetivo é determinar uma rota com menor tempo de percurso, no PML o objetivo é minimizar a soma dos tempos que os clientes aguardam por atendimento. O Problema da Mínima Latência também é referenciado na literatura como *Traveling Repairman Problem* (Tsitsiklis, 1992), *Delivery Man Problem* (Fischetti *et al.*, 1993), *School-bus Driver Problem* (Chaudhuri *et al.*, 2003) e também como Problema do Caixeiro Viajante com custos Cumulativos (Bianco *et al.*, 1993). Considerando-se espaços métricos gerais e também quando a estrutura subjacente é uma árvore com pesos nas arestas é possível mostrar que o PML é um problema NP-difícil (Sahni e Gonzalez, 1976; Sitters, 2002)

De acordo com o tipo de percurso a ser determinado, existem dois tipos de instâncias na literatura. No primeiro deles, o objetivo é determinar um caminho hamiltoniano com início no vértice zero (depósito). Já no segundo tipo, deve-se determinar um circuito hamiltoniano, com início e fim no depósito.

Aplicações do PML podem ser encontrados em sistemas de distribuição que focam na satisfação do cliente. Exemplos incluem serviços de entrega a domicílio (Méndez-Díaz *et al.*, 2008), atendimento emergencial como atendimento médico / serviço de reparo (Campbell *et al.*, 2008) e também relacionado a recuperação de dados em rede de computadores (Ezzine *et al.*, 2010).

Considere um dígrafo $G = (V, A, w)$, onde $V = \{0, \ldots, n\}$ é o conjunto de vértices e $A = \{(i, j) : i, j \in V, i \neq j\}$ é o conjunto de arcos sendo que para cada arco (i, j) está associado um tempo de percurso w_{ij}. O objetivo do Problema da Mínima Latência é determinar o circuito (ou caminho) hamiltoniano que minimize o tempo total de espera, $\sum_{i=0}^{n} t_i$, onde t_i representa o tempo de espera ou latência de um vértice $i \in V$. A latência é definida pelo tempo total para, a partir depósito, se alcançar o vértice i. No conjunto V, o vértice 0 representa o depósito e os demais vértices são os clientes a serem atendidos. Em algumas circunstâncias, além do tempo de percurso, considera-se um valor que corresponde ao tempo de atendimento dos clientes (Salehipour *et al.*, 2011).

Algumas características específicas do Problema da Mínima Latência fazem deste problema mais difícil de resolver e aproximar se comparado com o PCV, como afirmam Blum *et al.* (1994). Uma destas características que merece destaque é o caráter não local da função objetivo, ou seja, a inserção de um arco no início do circuito aumenta a latência de todos os clientes localizados após este arco na rota.

Na literatura há várias abordagens exatas para o PML. O modelo de programação linear inteira mista apresentado neste trabalho foi proposto por van Eijl (1995). Outras formulações podem ser vistas nos trabalhos de Méndez-Díaz *et al.* (2008); Bigras *et al.* (2008); Ezzine *et al.* (2010). Entre os trabalhos mais recentes destaca-se o de Abeledo *et al.* (2010) que utilizou uma abordagem capaz de resolver de forma exata instâncias com até 107 nós. Para isso foi utilizado um algoritmo de *Branch-and-Cut-and-Price* associado a uma formulação estendida, além de outros recursos.

Em relação às abordagens heurísticas, vários trabalhos recentes foram publicados. No trabalho de Salehipour *et al.* (2011), os autores apresentam um algoritmo combinando características das metaheurísticas GRASP (*Greedy Randomized Adaptive Search Procedure*), VNS (*Variable Neighborhood Search*) e VND (*Variable Neighborhood Descent*) para tratar o PML. Em Silva *et al.* (2011) foi proposta uma abordagem baseada em GRASP e ILS (*Iterated Local Search*) com busca local realizada através de uma heurística VND com ordem aleatória nas vizinhanças. Uma versão mais recente deste último trabalho foi publicada em Silva *et al.* (2012).

No trabalho proposto por Ngueveu *et al.* (2010) foi apresentada uma heurística baseada em algoritmos meméticos para tratar o Problema de Mínima Latência considerando uma frota de veículos (Problema de Roteamento de Veículos com custos Cumulativos). O algoritmo memético

(Neri *et al*., 2012; Merz, 2002), uma abordagem pertencente à classe dos algoritmos evolucionários, se baseia no processo de seleção natural e evolução de uma população. A abordagem apresentada neste trabalho é similar a esse algoritmo.

Na heurística de Ngueveu *et al*. (2010), as soluções que compõem a população inicial são criadas a partir da heurística do vizinho mais próximo, sendo que parte destas soluções são submetidas a um algoritmo de busca local. Para gerar uma nova população, a seleção para o cruzamento é realizado por torneio binário. Após o cruzamento, é verificado se a inversão da rota produz indivíduos de melhor qualidade. A melhor solução obtida após o cruzamento pode ser submetida a um procedimento de refinamento com uma determinada probabilidade (parâmetro do algoritmo). Para ser inserida na população da próxima geração, a nova solução deve atender a um critério de diversidade mínima em relação aos demais indivíduos desta população. As buscas locais são baseadas nos movimentos de reinserção, troca e 2-opt. Além da heurística, outra importante contribuição dos autores é um procedimento que melhora a eficiência da busca local, permitindo que o custo de uma solução vizinha possa ser recalculado em tempo constante.

Neste trabalho é apresentado um algoritmo memético que faz uso de duas das estruturas de vizinhança utilizadas por Ngueveu *et al*. (2010), 2-opt e reinserção. Além destas estratégias, um procedimento otimizado de reavaliação das soluções também é utilizado. As demais características são diferenciadas, o que possibilitou a obtenção de soluções de boa qualidade em um tempo computacional reduzido. Para dois diferentes grupos de instâncias da literatura, os testes preliminares deste trabalho apresentam resultados competitivos quando comparados com os resultados de Silva *et al*. (2012), que por sua vez, são melhores do que os obtidos pelo algoritmo de Ngueveu *et al*. (2010).

O restante deste trabalho está organizado da seguinte forma. Na Seção 2 é apresentada uma descrição formal do PML. Na Seção 3 é descrito o algoritmo memético avaliado neste trabalho. Os resultados preliminares são apresentados na seção 4 e conclusões e trabalhos futuros são apresentados na seção 5.

2 Modelo Matemático

O Problema de Mínima Latência pode ser definido a partir de um dígrado $G = (V, A, w)$, como descrito na seção anterior. A constante w_{ij} representa, para cada arco $(i, j) \in A$, o tempo de percurso entre os vértices i e j. Para este modelo, proposto por van Eijl (1995), considera-se que as rotas tem início e fim no vértice 0 (depósito), começando sempre no tempo 0.

Para a descrição do PML, duas variáveis são utilizadas. A variável binária x_{ij} indica se o arco $(i, j) \in A$ é utilizado na rota e a variável de tempo t_{ij} define o tempo de partida do nó i se o arco (i, j) tiver sido utilizado ($x_{ij} = 1$); $t_{ij} = 0$ caso contrário. Considerando C uma constante com valor suficientemente grande, o modelo de programação linear inteira mista é apresentado a seguir.

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

Sujeito a:

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

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

$$\sum_{i=1, i \neq j}^{n} t_{ij} + \sum_{i=0, i \neq j}^{n} w_{ij} \times x_{ij} = \sum_{k=0, k \neq j}^{n} t_{jk}, \qquad j \in V \setminus \{0\} \tag{4}$$

$$0 \leq t_{ij} \leq C x_{ij}, \qquad i \in V \setminus \{0\},\ j \in V \text{ e } i \neq j \tag{5}$$

$$x_{i,j} \in \{0, 1\}, \qquad i, j \in V \text{ e } i \neq j \tag{6}$$

O objetivo do PML, como visto função (1), é obter uma rota onde a soma dos tempos de partida seja minimizada. As restrições (2) e (3) garantem que cada nó do conjunto V seja visitado exatamente uma vez. As restrições (4), além de evitarem sub-rotas, definem que o tempo de partida do vértice j é igual ao tempo de partida do vértice i somado ao tempo de viagem w_{ij}, quando $x_{ij} = 1$. As restrições (5), para os casos onde $x_{ij} = 1$, definem um limite superior para t_{ij}. Se $x_{ij} = 0$, então esta restrição obriga que t_{ij} seja igual a zero. As restrições (6) definem o domínio das variáveis x_{ij}.

3 Heurísticas Propostas

O método proposto neste trabalho é um algoritmo memético, ou seja, um algoritmo genético que utiliza buscas locais para tentar melhorar a qualidade das soluções obtidas a cada geração. Para representar uma solução, o cromossomo foi codificado através de um vetor contendo uma permutação dos clientes, não incluindo o depósito. O algoritmo proposto utiliza uma população P contendo um conjunto de p cromossomos. O funcionamento da heurística proposta pode ser observado no pseudocódigo apresentado no Algoritmo 1, que recebe como parâmetro o tamanho p da população, o número máximo de iterações sem melhora ($maxIsm$), o percentual de mutação (mut) e a probabilidade de busca local (bl).

Nas linhas de 2 a 6 são obtidos os cromossomos que compõem a população inicial. De posse desta primeira geração, inicia-se o processo evolutivo que é executado até um determinado número de gerações sem atualização da melhor solução obtida pelo algoritmo. A cada iteração uma nova população P' é construída, sendo preservada a elite da população P. Para gerar novos indivíduos, é realizado um procedimento de cruzamento (linha 13) a partir de pares de soluções da população anterior (linhas 11 e 12). Após este processo, os indivíduos da nova população podem sofrer mutação (linhas 15 a 19) com determinada probabilidade $pmut$, que é proporcional ao número de iterações sem melhora (linha OPP). Um procedimento semelhante é utilizado para definir os cromossomos que serão submetidos à heurísticas de refinamento (linhas 20 a 23). Para completar a população mantendo uma diversidade mínima entre seus indivíduos, a cada geração são incluídas soluções geradas de forma aleatória na nova população. Nas próximas subseções são detalhandos cada um dos módulos que fazem parte desta abordagem.

3.1 Construção da população

A população inicial é criada a partir de indivíduos gerados de forma aleatória (linha 2) e, em seguida, aplicando-se a heurística de busca local 2-opt em ($bl \times p$) desses indivíduos (linhas

Algoritmo 1 Algoritmo Memético

```
1: Parametros: p, maxIsm, mut, bl
2: P ← população aleatória;
3: para i = 1 → |P| ∗ 0.2 faça
4:    S ← individuo qualquer de P
5:    2-opt(S)
6: fim para
7: ism = 0;
8: enquanto ism < maxIsm faça
9:    P' ← elite(P);
10:   para i = 1 → 0.75 × p faça
11:      pai1 ← individuo qualquer de elite(P)
12:      pai2 ← individuo qualquer de P \ elite(P)
13:      P' ← P' + cruzamento(pai1,pai2);
14:   fim para
15:   pmut ← minimo(mut × ism , 0.2)
16:   para i = 1 → p × pmut faça
17:      S ← individuo qualquer de P' \ melhorSolucao(P')
18:      mutacao(S)
19:   fim para
20:   para i = 1 → p × bl faça
21:      S ← individuo qualquer de P' \ elite(P')
22:      buscaLocal(S)
23:   fim para
24:   enquanto |P'| ≠ p faça
25:      P' ← P' + solucao aleatoria
26:   fim enquanto
27:   se ( custo(melhorSolucao(P')) = custo(melhorSolucao(P)) ) então
28:      ism ← ism + 1;
29:   senão
30:      ism ← 0;
31:   fim se
32:   P ← P'
33: fim enquanto
```

3-6). Essa população é armazenada em um arranjo P, que é mantido ordenado e dividido em três classes:

- Classe A (elite): Composta pelo melhores indivíduos e possui tamanho correspondente a 10% do tamanho de P.
- Classe B: São os indivíduos de P que não pertencem à elite e não pertencem à classe C (classe intermediária).
- Classe C: São os indivíduos com pior função objetivo e correspondem a 15% da população.

3.2 Criação da nova população

Após a etapa de construção, é realizado um processo iterativo que, a cada passo, cria uma nova geração da população. Essa nova população é obtida como representado na Figura 1. A nova população P' preserva os cromossomos associados à classe A da população corrente P. Existe a possibilidade destes indivíduos sofrerem um procedimento de mutação, exceto pelo melhor elemento da população que é totalmente preservado.

A seguir, são realizados cruzamentos de forma a inserir uma quantidade de indivíduos correspondente ao tamanho da classe B. Cada um desses cruzamentos envolve um indivíduo da classe A com um indivíduo das classes B ou C de P.

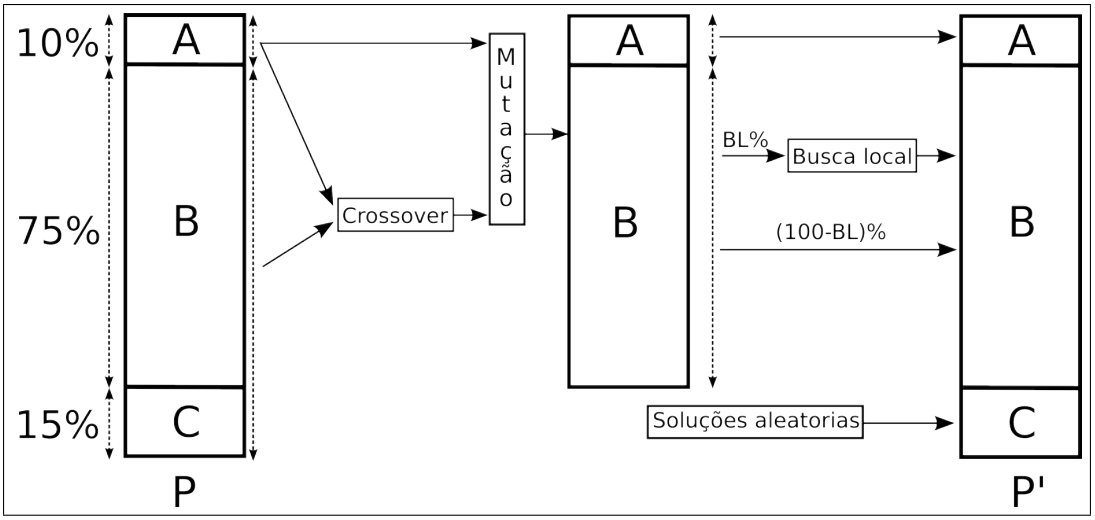

Figura 1: Etapas utilizadas para criar a nova população

Mais especificamente, dados dois pais p_1 e p_2, o cruzamento entre estes indivíduos é realizado da seguinte forma: São selecionados de forma aleatória dois pontos de corte, sendo o primeiro selecionado na primeira metade de cada indivíduo e o segundo na segunda metade. Tais pontos de corte dividem cada pai em três trechos. Então, são gerados dois filhos f_1 e f_2. O filho f_1 é gerado a partir de dois trechos do pai p_1, o primeiro e o terceiro, e complementado com o segundo trecho do pai p_2. De forma similar, o filho f_2 é gerado a partir de dois trechos do pai p_2, o primeiro e o terceiro, e complementado com o segundo trecho do pai p_1. A Figura 2 ilustra esse processo. Note que esse procedimento pode gerar indivíduos inválidos já que alguns vértices presentes em um trecho de um pai pode também estar presente no trecho utilizado do outro pai. Para contornar esse problema, as posições do trecho central que geram conflitos são deixadas em branco e preenchidas posteriormente de acordo com a ordem em que os vértices ausentes aparecem no pai do qual o filho herdou dois trechos. A Figura 3 exibe um exemplo de cruzamento que apresenta esse problema.

Observe que a ordem dos vértices marcados de vermelho no filho f_1 (vértices 4 e 7) foi determinada com base na sequencia do pai p_1. O filho f_2 apresenta uma situação semelhante e, nesse caso, a ordem dos vértices marcados de vermelho foi definida com base em p_2.

Vale mencionar que, apesar de serem gerados dois filhos em cada cruzamento, apenas o filho mais adaptado (ou seja, o que tiver melhor função objetivo) é inserido na nova população.

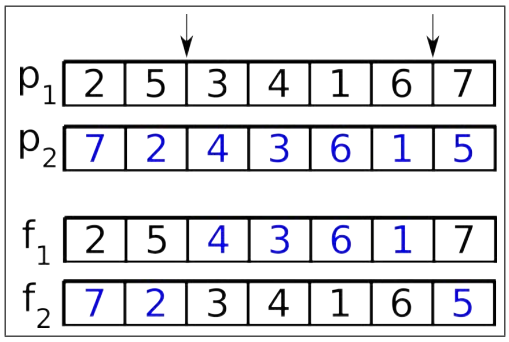

Figura 2: Exemplo de cruzamento entre dois pais

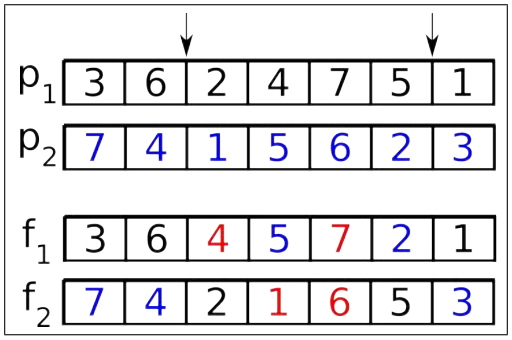

Figura 3: Exemplo de cruzamento onde vértices repetidos são inseridos em um mesmo cromossomo

Após a etapa de cruzamento, é aplicado um procedimento de mutação em $pmut \times p$ dos indivíduos, que são escolhidos de forma aleatória. O valor de $pmut$ é obtido multiplicando-se o valor de mut (parâmetro do algoritmo) pelo número de iterações sem melhora, sendo limitado a no máximo $0,2$, ou seja, no máximo 20% dos indivíduos sofrem mutação.

Dado um cromossomo S, a mutação em S é realizada utilizando 5 operações de *swap*, ou seja, são sorteados 5 pares de posições do cromossomo e, para cada par i, j sorteado, os vértices que estiverem nas posições i e j são intercambiados.

Realizada a mutação, na linha 22, $bl \times p$ dos indivíduos são submetidos a um procedimento de busca local, que será descrito na seção 3.3. Finalmente, o restante da população é preenchido com indivíduos gerados de forma aleatória (linha 25).

Note que a divisão da população em classes, uma estratégia já empregada na literatura (Breslau *et al.*, 2011), é realizada para garantir que os melhores indivíduos (classe A) sejam preservados e para proporcionar uma diversidade da população.

3.3 Buscas Locais

Para a etapa de busca local, o algoritmo proposto utilizou duas estruturas de vizinhança amplamente descritas na literatura: as estruturas 2-opt e reinserção. Na primeira, duas arestas não adjacentes são removidas e outras duas são adicionadas de modo a construir uma nova solução válida. Já na reinserção, um nó é simplesmente realocado para uma nova posição do percurso.

As duas estruturas de vizinhança são utilizadas em sequencia, ou seja, dada uma solução S, primeiro é aplicada uma busca do tipo 2-opt em S, em seguida, é aplicada a reinserção na solução melhorada pelo 2-opt. Em ambas as abordagens foi utilizada a estratégia de melhor aprimorante para percorrer o espaço de busca.

A Figura 4 apresenta um exemplo de solução vizinha gerada a partir da busca local com vizinhança do tipo reinserção. Note que, para se calcular o custo de uma solução vizinha, é necessário remover 3 arcos (no caso do exemplo, os arcos $(2,3)$, $(3,4)$ e $(5,6)$), inserir outros 3 novos arcos (no caso do exemplo, os arcos $(2,4)$, $(5,3)$ e $(3,6)$) e, então, calcular o custo gerado a partir do deslocamento de um trecho da solução (nesse caso, o trecho formado pelo arco $(4,5)$ foi deslocado uma unidade para a esquerda). Note que, apesar do cálculo do custo das arestas removidas e inseridas poder ser realizado em tempo constante, uma estratégia trivial para recalcular o custo do trecho deslocado teria complexidade linear em relação ao número $|V|$ de vértices da solução, já que o tamanho desse trecho pode ser, no pior caso, igual a $|V|-1$.

Um exemplo de solução vizinha gerada a partir da busca local 2-opt é apresentado na Figura 5: Primeiro, removem-se 2 arcos (nesse caso, os arcos $(2,3)$ e $(6,7)$), inserem-se outros 2 arcos ($(2,6)$ e $(3,7)$) e, então, inverte-se o sentido de um trecho da solução (no exemplo dado, o trecho compreendido entre os vértices 3 e 6 deverá ser invertido). De forma semelhante à da reinserção, uma estratégia trivial para recalcular o custo da solução vizinha teria tempo linear em relação a $|V|$ visto que é necessário recalcular o custo associado aos arcos no trecho invertido e esse trecho possui tamanho, no pior caso, igual a $|V|-1$.

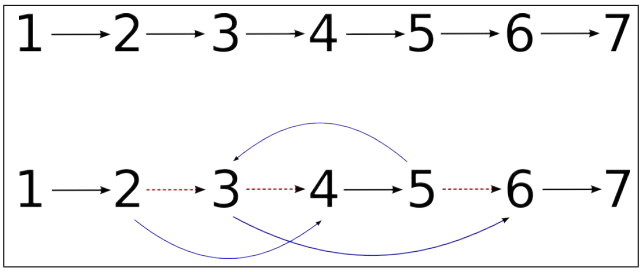

Figura 4: Busca local com vizinhança reinserção

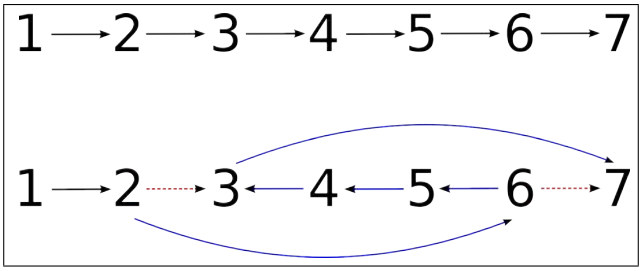

Figura 5: Busca local com vizinhança 2-opt

Como as duas estruturas de vizinhança possuem tamanho $O(|V|^2)$, o uso de uma estratégia trivial para recalcular o custo da função objetivo de soluções vizinhas tornaria a complexidade de cada iteração da busca local $O(|V|^2) \times O(|V|) = O(|V|^3)$ (Ngueveu *et al.*, 2010). Neste trabalho, foi utilizada uma estratégia semelhante à utilizada por Silva *et al.* (2012), que consiste em pré-computar, antes de cada iteração da busca local, uma tabela que permite obter os custos associados a cada possível rota deslocada ou invertida em tempo constante. Como a tabela utilizada pode ser calculada em tempo $O(|V|^2)$ Silva *et al.* (2012), a complexidade de cada iteração

da busca local é reduzida para $O(|V|^2) + O(|V|^2) \times O(1) = O(|V|^2)$.

4 Resultados

O algoritmo genético proposto por esse trabalho foi implementado na linguagem C++, compilado com o g++ 4.6 e executado em um computador com as seguintes características: processador Intel Core i5-2400, 8Gb de memória RAM, Sistema Operacional GNU/Linux Ubuntu 11.10. Foram realizadas duas baterias de testes envolvendo dois grupos distintos de instâncias disponibilizadas em literatura por Abeledo *et al*. (2010) e Salehipour *et al*. (2011). As instâncias do primeiro grupo possuem de 42 a 107 vértices e as soluções para esse grupo consistem em circuitos hamiltonianos com início e fim na origem (vértice 0), enquanto que as instâncias do segundo grupo possuem como solução um caminho hamiltoniano com início na origem.

O método proposto foi comparado com o método *GILS-RVND* Silva *et al*. (2012) que, pelo que se sabe, é o método mais eficiente para a solução do PML descrito em literatura.

Para determinar a melhor combinação de parâmetros a serem utilizados pelo algoritmo foi realizada uma avaliação estatística. Para esta avaliação foram definidos os valores máximo e mínimo para cada parâmetro. O tamanho da população foi definido em função do número de vértices da instância, os valores limites para esse parâmetro são 50% da número de vértices e o máximo 150% deste valor. Para o número de iterações sem melhora, 15 foi o limite inferior e 100 o limite superior. Já para o percentual de mutação, foram definidos como limites 0% e 2%. Por fim, para a probabilidade de busca local, o parâmetro foi avaliado no intervalo $[20\%, 50\%]$. Dados estes limites, 25 combinações distintas de parâmetros foram definidas e avaliadas com objetivo de determinar a melhor combinação possível. Para cada uma destas combinações o algoritmo foi executado 30 vezes para cada uma das instâncias. Após a análise dos resultados, foi possível verificar a robustez do algoritmo. Nenhuma das configurações obteve resultados estatisticamente melhor entre as combinações avaliadas. Considerando a qualidade média das soluções e o tempo médio de CPU consumido por cada combinação de parâmetros, optou-se por utilizar nos testes finais o tamanho da população igual ao número de vértices da instância, número de iterações sem melhora igual a 58, percentual de mutação de 2% e probabilidade de busca local correspondendo a 20%. Para esta combinação foram realizadas 30 execuções para cada uma das instâncias e os resultados são apresentados a seguir.

Os resultados obtidos para as instâncias de Abeledo *et al*. (2010) e Salehipour *et al*. (2011) são apresentados, respectivamente, nas Tabela 1 e 2. A coluna **Prob.** indica o nome da instância testada, **Mel. Sol.** indica a melhor solução obtida pelo algoritmo considerando todas as execuções. A coluna **Méd. Sol.** indica a média das soluções obtidas e a coluna **Tempo** indica a média do tempo (em segundos) das 30 execuções. Para cada instância, o menor valor de função objetivo obtido pelos algoritmos é apresentada em negrito.

Conforme pode ser observado na Tabela 1, em todos as instâncias de Abeledo *et al*. (2010) o algoritmo proposto obteve a solução ótima em pelo menos uma das execuções. Nos casos onde o valor de **Méd. Sol.** está em negrito, o algoritmo consegue obter a solução ótima em todas as 30 execuções. É possível comparar também o tempo de execução dos algoritmos, onde em alguns casos, o algoritmo proposto por esse trabalho conseguiu ser até $5,7$ vezes mais rápido do que o método *GILS-RVND*.

No grupo de instâncias de Salehipour *et al*. (2011) (Tabela 2), o algoritmo também obteve a melhor solução conhecida em pelo menos uma das execuções. O *speed up* diminuiu à medida que o tamanho das instâncias aumentou, sendo que o tempo de execução das três últimas instâncias foi maior no algoritmo memético do que no *GILS-RVND*. Porém, note que para a última instância a melhor solução obtida pelo método proposto apresentou função objetivo mais competitiva do que a melhor solução da literatura.

	Algoritmo Memético			GILS-RVND		
Prob.	Mel. Sol.	Méd. Sol.	Tempo	Mel. Sol.	Méd. Sol.	Tempo
dantzig42	**12528**	12529.07	0.06	**12528**	**12528.00**	0.16
swiss42	**22327**	**22327.00**	0.05	**22327**	**22327.00**	0.16
att48	**209320**	209481.00	0.09	**209320**	**209320.00**	0.32
gr48	**102378**	**102378.00**	0.10	**102378**	**102378.00**	0.33
hk48	**247926**	**247926.00**	0.09	**247926**	**247926.00**	0.30
eil51	**10178**	10211.20	0.14	**10178**	**10178.00**	0.49
berlin52	**143721**	**143721.00**	0.12	**143721**	**143721.00**	0.46
brazil58	**512361**	**512361.00**	0.18	**512361**	**512361.00**	0.78
st70	**20557**	20564.00	0.37	**20557**	**20557.00**	1.65
eil76	**17976**	18030.60	0.68	**17976**	**17976.00**	2.64
pr76	**3455242**	3460609.00	0.67	**3455242**	**3455242.00**	2.31
gr96	**2097170**	2102120.87	1.47	**2097170**	**2097171.00**	2.41
rat99	**57986**	58115.10	2.05	**57986**	**57986.00**	11.27
kroA100	**983128**	983133.87	1.65	**983128**	**983128.00**	8.59
kroB100	**986008**	986118.53	1.61	**986008**	**986008.00**	9.21
kroC100	**961234**	**961324.00**	1.65	**961324**	**961324.00**	8.17
kroD100	**976965**	978066.67	1.69	**976965**	**976865.00**	8.46
kroE100	**971266**	**971266.00**	1.59	**971266**	**971266.00**	8.31
rd100	**340047**	340475.37	1.86	**340047**	**340047.00**	8.52
eil101	**27513**	27595.30	2.66	**27513**	**27513.00**	12.76
lin105	**603910**	**603910.00**	1.98	**603910**	**603910.00**	8.42
pr107	**2026626**	2026904.60	2.05	**2026626**	**2026626.00**	10.89

Tabela 1: Resultados computacionais para o grupo de instâncias de Abeledo *et.al.*

	Algoritmo Memético			GILS-RVND		
Prob.	Mel. Sol.	Méd. Sol.	Tempo	Mel. Sol.	Méd. Sol.	Tempo
st70	**19215**	**19215.00**	0.35	**19215**	**19215.00**	1.51
rat99	**54984**	55073.87	2.05	**54984**	**54984.00**	9.47
kroD100	**949594**	**949594.00**	1.68	**949594**	**949594.00**	6.90
lin105	**585823**	**585823.00**	1.72	**585823**	**585823.00**	6.19
pr107	**1980767**	**1980767.00**	2.18	**1980767**	**1980767.00**	8.13
rat195	**210191**	210482.07	37.33	**210191**	210335.90	75.56
pr226	**7100308**	**7100308.00**	41.47	**7100308**	**7100308.00**	59.05
lin318	**5560679**	5579900.00	284.40	**5560679**	5569819.50	220.59
pr439	**17688561**	17742120.97	1252.66	**17688561**	17734922.00	553.74
att532	**5577965**	5598682.40	4079.65	5581240	5597866.80	1792.61

Tabela 2: Resultados computacionais para o grupo de instâncias de Salehipour *et.al.*

5 Conclusões e Trabalhos Futuros

O algoritmo proposto por este artigo consiste na aplicação de uma metaheurística baseada em um Algoritmo Genético juntamente com uma busca local para o Problema da Mínima Latência. O objetivo é encontrar uma ordem de visitação cuja soma dos tempos de espera dos clientes seja minimizada.

O algoritmo foi aplicado em diversas instâncias disponíveis em literatura, tendo tamanho que varia entre 42 e 532 vértices. Conforme mencionado na Seção 4, as soluções apresentadas foram competitivas em relação às soluções da literatura. Em todos os casos, a melhor solução encontrada pelo algoritmo é a melhor solução conhecida em literatura. Além disso, o algoritmo foi mais rápido do que o método proposto por Silva *et al*. (2012) em 29 das 32 instâncias, obtendo um *speed-up* de até $5,7$ vezes.

Como trabalhos futuros, pretende-se melhorar o desempenho do algoritmo principalmente nas instâncias maiores. Além disso, pretende-se desenvolver uma versão paralela do método com o objetivo de diminuir o tempo de processamento.

Referências

Abeledo, H. G., Fukasawa, R., Pessoa, A. A. e Uchoa, E. The time dependent traveling salesman problem: Polyhedra and branch-cut-and-price algorithm. Festa, P. (Ed.), *SEA*, volume 6049 of *Lecture Notes in Computer Science*, p. 202–213. Springer. ISBN 978-3-642-13192-9, 2010.

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, n. 4, p. 685 – 699.

Blum, A., Chalasani, P., Coppersmith, D., Pulleyblank, 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. ISBN 0-89791-663-8. doi: 10.1145/195058.195125. URL `http://doi.acm.org/10.1145/195058.195125`, 1994.

Breslau, L., Diakonikolas, I., Duffield, N., Gu, Y., Hajiaghayi, M., a nd Howard Karloff, D. S. J., Resende, M. G. C. e Sen, S. Disjoint-path facility location: Theory and practice. MÃ$\frac{1}{4}$ller-Hannemann, M. e Werneck, R. F. F. (Eds.), *ALENEX*, p. 60–74. SIAM, 2011.

Campbell, A. M., Vandenbussche, D. e Hermann, W. (2008), Routing for relief efforts. *Transportation Science*, v. 42, n. 2, p. 127–145.

Chaudhuri, K., Godfrey, B., Rao, S. e Talwar, K. Paths, trees, and minimum latency tours. *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '03, p. 36–, Washington, DC, USA. IEEE Computer Society. ISBN 0-7695-2040-5. URL `http://dl.acm.org/citation.cfm?id=946243.946316`, 2003.

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

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

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

Merz, P. A comparison of memetic recombination operators for the traveling salesman problem. *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, p. 472–479, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. ISBN 1-55860-878-8. URL http://dl.acm.org/citation.cfm?id=646205.682328, 2002.

Neri, F., Cotta, C. e Moscato, P. *Handbook of Memetic Algorithms*. Studies in Computational Intelligence. Springer. ISBN 9783642232466. URL http://books.google.com.br/books?id=uop6UvKu8q4C, 2012.

Ngueveu, S. U., Prins, C. e Wolfler Calvo, R. (2010), An effective memetic algorithm for the cumulative capacitated vehicle routing problem. *Comput. Oper. Res.*, v. 37, n. 11, p. 1877–1885.

Sahni, S. e Gonzalez, T. F. (1976), P-complete approximation problems. *J. 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*, v. 9, n. 2, p. 189–209.

Silva, M., Subramanian, A. e Ochi, L. S. Uma heuristica baseada em grasp e iterated local search para o problema da minima latencia. *In Proc. 43th Simpósio Brasileiro de Pesquisa Operacional*, p. 1813–1823, 2011.

Silva, M. M., Subramanian, A., Vidal, T. e Ochi, L. S. (2012), A simple and effective metaheuristic for the minimum latency problem. *European Journal of Operational Research*, v. , n. 0, p. –.

Sitters, R. The minimum latency problem is np-hard for weighted trees. *In Proc. 9th Integer Programming and Combinatorial Optimization Conference*, p. 230–239, 2002.

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 polyhedral approach to the delivery man problem. Relatório técnico, Eindhoven University of Technology, 1995.