Uma Heurística Paralela Eficiente para o Problema da Mínima Latência

Eyder Rios
Cristina Boeres
Instituto de Computação - Universidade Federal Fluminense
Rua Passo da Pátria, 156, Bloco E, 3ºandar, CEP 24210-240, Niterói, RJ
{eyder, boeres}@ic.uff.br

RESUMO

O Problema da Mínima Latência (PML) é uma variante do Problema do Caixeiro Viajante (PCV) cujo objetivo é minimizar o tempo global de chegada aos vértices, e não o tempo de deslocamento como ocorre no problema original. Este artigo introduz uma heurística paralela simples e eficiente para resolução do PML que conjuga conceitos de diferentes meta-heurísticas em um ambiente paralelo. A combinação destes conceitos, juntamente com um mecanismo de cooperação entre tarefas paralelas, permitiu o desenvolvimento de um algoritmo cuja eficácia foi comprovada por experimentos realizados em 173 instâncias com até 1.000 clientes cada. De fato, o método proposto conseguiu equiparar ou suplantar os melhores resultados da literatura, apresentando 25 novas melhores soluções para o problema. Os tempos computacionais obtidos foram bastante competitivos, alcançando *speedups* médios linear ou superlinear nos experimentos realizados.

PALAVRAS CHAVE: Problema da Mínima Latência, Heurística Paralela, GRASP, ILS, RVND

Áreas Principais: Otimização Combinatória, Processamento Paralelo

ABSTRACT

The Minimum Latency Problem (MLP) is a variant of the classical Travelling Salesman Problem (TSP) whose objective is to minimize the overall arrival time to the vertices, instead travel time as in the original problem. This paper introduces a simple parallel heuristic to solve the MLP that merges concepts of differents metaheuristics on a parallel environment. The combination of these elements, along with a cooperation mechanism carried out by parallel tasks, enabled to develop an algorithm whose effectiveness has been proven by experiments performed on 173 instances with up to 1,000 customers each. In fact, the method were able to match or improve the best results of the literature, arising 25 new best solutions for the problem. The computacional times obtained were very competitive, achieving average linear or superlinear speedups for performed experiments.

KEYWORDS: Minimum Latency Problem, Parallel Heuristic, GRASP, ILS, RVND

Main areas: Combinatorial Optimization, Parallel Processing

1 Introdução

Em problemas de roteamento de veículos, o tempo ou a distância percorrida são as principais métricas utilizadas na avaliação do custo de deslocamento entre os vértices. Já em problemas de redes, a métrica dominante é a latência, que está associada ao tempo de espera até a chegada a um vértice a partir de um ponto de partida. O Problema da Mínima Latência, do inglês *Minimum Latency Problem*, é uma variante do Problema do Caixeiro Viajante (PCV), cujo objetivo é minimizar o tempo de chegada aos vértices e não o tempo de deslocamento ou a distância percorrida como ocorre no problema original.

O Problema da Mínima Latência (PML) pode ser definido como um grafo direcionado $G = (V, E)$, onde $V = \{0, 1, \ldots, n\}$ representa o conjunto de vértices e $E = \{(i, j) : i, j \in V, i \neq j\}$ o conjunto de arcos que conectam os vértices. Cada arco (i, j) possui um tempo de deslocamento associado igual a $t(i, j)$. O vértice 0 representa o ponto de partida (depósito) e os demais vértices os clientes a serem visitados. A latência de um cliente $i \in V$, denotada por $l(i)$, é definida como o tempo de deslocamento entre o depósito e o vértice i. O objetivo do PML é, partindo do depósito, determinar o circuito Hamiltoniano s que minimiza a latência total, expressada por $L(s) = \sum_{i=0}^{n} l(i)$.

O PML é também referenciado na literatura como *Travelling Repairman Problem*, *Delivery Man Problem*, *Cumulative Traveling Salesman Problem* e *School Bus Driver Problem*. Alguns trabalhos não consideram o retorno ao depósito como uma restrição do problema. Porém, esta discrepância pode ser facilmente resolvida adicionando-se um cliente fictício $k = n + 1 \in V$ tal que $t(0, k) = t(k, 0) = 0$.

Apesar de sua formulação simples, o PML é um problema complexo. De fato, foi provado que este problema pertence à classe NP-Difícil quando os vértices residem em espaços métricos gerais ou em planos euclidianos [Sahni e Gonzalez (1976); Arora e Karakostas (2003)] e também quando o espaço considerado é induzido por uma árvore ponderada [Sitters (2006)]. Além disso, a despeito de sua similaridade com PCV, este problema possui a reputação de ter resolução computacionalmente mais difícil que o primeiro. Como ressaltado por vários pesquisadores, do ponto de vista computacional o PML apresenta características que o distinguem do PCV. Uma delas é que pequenas alterações locais na configuração dos dados de entrada podem conduzir a mudanças globais significativas na estrutura da solução final [Blum *et al.* (1994)]. Outra característica marcante é a natureza não local da função objetivo, onde a simples inserção ou alteração de posição de um arco do circuito afeta a latência dos vértices posteriores, como ressaltado em [Arora e Karakostas (2003)].

Aplicações do PML destacam-se em diversos contextos. Em sistemas de distribuição por exemplo, o problema pode ser aplicado para priorizar a satisfação do usuário em detrimento do custo de deslocamento, já que busca minimizar o tempo de espera do cliente. Outras aplicações podem ser encontradas no processamento de busca de informações em redes de computadores, na recuperação de dados em dispositivos de armazenamento em disco e no escalonamento de tarefas [Salehipour *et al.* (2011); Ezzine *et al.* (2010); Angel-Bello *et al.* (2013)].

Este trabalho introduz uma heurística híbrida paralela para a resolução do Problema da Mínima Latência. A hibridização do algoritmo decorre da combinação de elementos de GRASP (*Greedy Randomized Adaptive Search Procedure*), ILS (*Iterated Local Search*) e RVND (*Variable Neighborhood Descent with Random neighborhood ordering*) [Gendreau e Potvin (2010)], além de uma técnica de avaliação de movimentos que permite o cálculo do custo da solução em tempo $O(1)$ [Silva *et al.* (2012)]. Todos estes elementos, aliados a estruturas de dados eficientes e a um ambiente *multithreaded*, permitiram o desenvolvimento de um algoritmo paralelo simples que apresenta desempenho e robustez

superior a outros trabalhos da literatura. Os experimentos computacionais realizados apresentaram resultados que introduzem novas melhores soluções, além de uma redução significativa do tempo de execução, especialmente para instâncias de grande porte com até 1.000 clientes. O levantamento bibliográfico realizado para a produção deste trabalho sugere que esta é a primeira heurística paralela para o PML.

O texto a seguir está organizado como se segue. Na seção 2 são apresentados outros trabalhos relacionados ao PML. Na seção 3 é descrito o algoritmo proposto. Os resultados dos experimentos computacionais são detalhados e analisados na Seção 4. Finalmente, na Seção 5, são apresentadas as considerações finais e propostas de trabalhos futuros.

2 Trabalhos Relacionados

Várias abordagens para o Problema da Mínima Latência podem ser encontrados na literatura. A maior parte destes trabalhos pode ser classificada em três diferentes grupos, de acordo com a abordagem empregada na resolução do problema: métodos aproximativos, exatos e heurísticos.

Uma limitação marcante dos trabalhos baseados em métodos exatos é que os experimentos computacionais realizados utilizam instâncias de pequenas dimensões, de algumas dezenas até pouco mais de uma centena de clientes. Isto ocorre porque a resolução exata de instâncias de médio e grande porte do PML envolve um tempo computacional proibitivo. Neste contexto, os métodos heurísticos surgem como uma alternativa interessante para obtenção de soluções de boa qualidade, eventualmente ótimas, a um custo computacional aceitável. No entanto, poucas abordagens heurísticas para o PML são encontradas na literatura. Um algoritmo baseado em Busca Tabu que emprega múltiplas estruturas de vizinhança foi desenvolvido em [Dewilde *et al.* (2013)] para resolver PML com coleta de prêmios. [Salehipour *et al.* (2011)] introduziram um método inspirado em GRASP, VND (*Variable Neighborhood Descent*) e VNS (*Variable Neighborhood Search*) aliado a um procedimento de perturbação baseado em relocações aleatórias e restrições de vizinhança, que resolveu instâncias de até 1.000 clientes. Os resultados deste último trabalho foram todos suplantados por [Silva *et al.* (2012)] com seu algoritmo baseado em GRASP, ILS, RVND combinado com um mecanismo de perturbação. Este último estudo ainda introduziu um novo método para avaliação de movimentos em tempo constante.

Outras abordagens encontradas foram desenvolvidas originalmente para o Problema de Roteamento de Veículos Cumulativo (PRVC), do inglês *Cumulative Vehicle Routing Problem*. Porém, tais aplicações podem ser mapeadas para o PML quando apenas um veículo é considerado. Neste contexto, o trabalho de [Ngueveu *et al.* (2010)] propõe um algoritmo evolutivo para o problema que introduz uma nova técnica de avaliação de movimentos de tempo linear para alguns tipos especiais de vizinhanças. Em [Chen *et al.* (2012)] uma heurística baseada em ILS resolveu instâncias com até 199 clientes.

Apesar dos métodos heurísticos produzirem soluções de boa qualidade a um custo computacional reduzido, o esforço empenhado para a resolução de instâncias de médio e grande porte ainda constitui um desafio. De fato, tal esforço acaba por estabelecer um limite para a intensidade da busca realizada pelos métodos heurísticos sequenciais. Uma forma de ampliar a busca pode ser obtido por meio da implementação de algoritmos paralelos, que incrementam o poder computacional sem sacrificar demasiadamente o tempo de execução. Este parece ser um campo ainda inexplorado pelos métodos que buscam resolver o PML, uma vez que durante a pesquisa realizada para a produção deste trabalho, não foi encontrada nenhuma implementação heurística paralela para o problema.

Entretanto, a eficácia da abordagem paralela pode ser comprovada por trabalhos como os de [Subramanian *et al.* (2010)] e [Coelho *et al.* (2012)]. No primeiro, um algoritmo paralelo baseado em ILS e RVND foi executado em um ambiente *multi-core* de larga

escala. O método obteve novas soluções para o Problema de Roteamento de Veículos (PRV) com Coleta e Entrega Simultâneas, além de apresentar um bom comportamento frente à variação de escalabilidade do *hardware*. Por sua vez, o segundo trabalho investiu em uma estratégia inspirada em VNS e VND para resolver o PRV com Entregas e Coletas Seletivas utilizando arquitetura GPU (*Graphics Processing Unit*).O resultado foi a obtenção de novas melhores soluções para o problema, além da redução do tempo de execução em mais de 16 vezes quando comparado com a versão sequencial correspondente. Um outro trabalho interessante, desenvolvido por [Araújo *et al.* (2007)], estudou o comportamento de quatro diferentes estratégias de paralelização de um método embasado em GRASP e ILS para a programação de tabelas de torneios esportivos. Os experimentos realizados destacaram os benefícios do compartilhamento de informações numa estrutura hierárquica de tarefas face aos custos de comunicação envolvidos em ambientes paralelos heterogêneos de larga escala.

3 Algoritmo Proposto

A abordagem aqui proposta foi baseada na heurística sequencial elaborada por [Silva *et al.* (2012)] e introduz um algoritmo simples que explora o paralelismo intrínseco dos processadores modernos presentes em seus núcleos de processamento (*cores*). Além disso, a implementação possui um mecanismo de cooperação que compartilha soluções entre as tarefas paralelas, o qual contribui para uma rápida convergência para melhores resultados.

O algoritmo paralelo proposto, PGILS-SG, é composto por dois tipos de tarefas: uma tarefa mestre, que gerencia o compartilhamento de informações e controla a execução das demais; e tarefas trabalhadoras, que realizam efetivamente a busca por soluções. A tarefa mestre detém a melhor solução encontrada pela aplicação paralela (solução global) com as demais tarefas trabalhadoras. Durante a execução, a solução global é confrontada periodicamente com a solução corrente obtida por uma tarefa trabalhadora, quando pode ser atualizada caso apresente pior custo. Nesta mesma oportunidade, a tarefa trabalhadora pode ter sua melhor solução corrente atualizada pela solução global caso esta última apresente melhor qualidade. Nenhuma atualização ocorre quando as soluções confrontadas apresentam o mesmo custo. Na proposta atual, a solução global é compartilhada via memória principal.

O Algoritmo 1 descreve o procedimento executado pelas tarefas trabalhadoras. Cada tarefa mantém sua melhor solução corrente, s°, que é inicializada na linha 2. O laço mais externo (linhas 3-23) descreve o arcabouço de uma heurística GRASP, com número de iterações controlado pelo parâmetro G_{trb}. Uma solução s' é então criada usando o procedimento de construção definido por [Feo e Resende (1995)], cujo grau de aleatoriedade α é selecionado randomicamente do conjunto R, um parâmetro de entrada do algoritmo (linhas 6-7). A solução s' criada no passo anterior é então usada como solução inicial para um mecanismo similar ao ILS, cujo número de iterações é determinado pelo parâmetro de entrada I_{max} (linhas 8-18). A linha 12 mostra a chamada para um procedimento RVND que realiza uma busca exaustiva em torno de s' utilizando cinco diferentes estruturas de vizinhança, exatamente como descrito em [Silva *et al.* (2012)]. Após a busca local, a solução resultante s' é avaliada em relação a s, a melhor solução do procedimento ILS (linha 13). Caso s' apresente melhor custo, s é atualizada (linha 14) e o contador de iterações do ILS é reinicializado (linha 15). Na sequência, a solução s é perturbada para gerar uma nova solução para a próxima iteração (linha 17). O laço interno persiste até que I_{max} iterações sem atualização de s sejam atingidas. Finalizada as iterações ILS, a solução s é comparada com a melhor solução corrente da tarefa, s°, que é atualizada caso esta apresente pior custo (linhas 19-21). Finalmente, a melhor solução corrente da tarefa é confrontada com a solução global s^{*}, mantida pela tarefa mestre (linha 22). Durante este processo, a solução global

pode ser atualizada por s°, caso esta apresente melhor custo. A função $compartilha(\cdot)$ sempre retorna a solução global s^*, que pode ser a própria solução corrente da tarefa. Os procedimentos de construção (linha 7), RVND (linha 12) e perturbação (linha 17) são os mesmos definidos por [Silva *et al.* (2012)].

```
1  procedimento TGILS-SG(R, G_trb, I_max)
2      f(s°) ← ∞;
3      g ← 0;
4      enquanto g < G_trb faça
5          g ← g + 1;
6          α ←valor selecionado aleatoriamente de R;
7          s' ← construtor(α);
8          s ← s';
9          i ← 0;
10         enquanto i < I_max faça
11             i ← i + 1;
12             s' ← RVND(s');
13             se f(s') < f(s) então
14                 s ← s';
15                 i ← 0;
16             fim
17             s' ← perturb(s);
18         fim
19         se f(s) < f(s°) então
20             s° ← s;
21         fim
22         s° ← compartilha(s°);
23     fim
24 fim
```

Algoritmo 1: TGILS-SG: pseudocódigo do algoritmo executado pelas tarefas trabalhadoras.

Vale ressaltar alguns aspectos importantes em relação ao Algoritmo 1. Apesar do trecho relativo ao ILS (linhas 8-18) ter suas iterações limitadas pelo parâmetro I_{max}, este procedimento frequentemente executa uma quantidade maior de iterações, dependendo do número de vezes que a solução s é atualizada. Este comportamento é interessante porque o algoritmo prolonga autonomicamente sua execução à medida em que melhores soluções vão sendo encontradas. O mesmo ocorre internamente com o procedimento RVND (linha 12), que tem sua busca reinicializada sempre que consegue incrementar sua solução corrente. Um outro aspecto relevante é que a tarefa trabalhadora inicia sua busca sem utilizar a solução corrente como referência inicial. A ideia por trás desta conduta foi a de preservar ao máximo as características do algoritmo sequencial original, tendo em vista estudar o impacto da divisão de iterações e do compartilhamento de soluções. De fato, com exceção do código referente ao compartilhamento de solução (linha 22), o algoritmo executado pelas tarefas trabalhadoras é o mesmo da heurística sequencial.

O Algoritmo 2 apresenta o pseudocódigo da tarefa mestre, que mantém a melhor solução global do algoritmo paralelo, representada por s^*. A tarefa começa inicializando a solução global (linha 1). Em seguida, as iterações determinadas pelo parâmetro de entrada G_{max} são divididas em quotas a serem distribuídas entre p tarefas (linha 3). Estas quotas irão determinar o número de iterações realizadas pelo procedimento GRASP embutido nas tarefas trabalhadoras. Eventuais iterações remanescentes da divisão em

quotas são preservadas para posterior distribuição (linha 4). O laço seguinte realiza a criação das tarefas trabalhadoras (linhas 5-12). A quota de iterações GRASP de cada tarefa é atribuída inicialmente para *iters* (linha 6). Caso existam iterações remanescentes, elas são distribuídas entre as primeiras tarefas lançadas (linhas 7-10). Cada tarefa trabalhadora é então lançada recebendo o conjunto de valores α, definido pelo parâmetro de entrada R, e o número de iterações GRASP e ILS a serem executadas, determinados por *iters* e I_{max}, respectivamente. Após disparar as tarefas em paralelo, o procedimento aguarda que todas as tarefas trabalhadoras terminem suas respectivas execuções (linha 13) e, finalmente, retorna a melhor solução global encontrada (linha 14).

O número de tarefas da aplicação paralela é controlado por p. Tipicamente, este parâmetro de entrada é definido para o número de núcleos de processamento existentes no ambiente alvo, visando explorar todos os recursos paralelos disponíveis. Para efeitos de análise de desempenho, é importante ressaltar que ao executar o algoritmo, $p + 1$ tarefas são criadas: 1 tarefa mestre e p tarefas trabalhadoras.

```
1  procedimento PGILS-SG(p, R, G_max, I_max)
2      f(s*) ← ∞;
3      quota ← G_max div p;
4      extra ← G_max mod p;
5      para t ← 1, ..., p faça
6          iters ← quota;
7          se extra > 0 então
8              iters ← iters + 1;
9              extra ← extra − 1;
10         fim
11         TGILS-SG (R, iters, I_max);
12     fim
13     Aguarda até que todas as instâncias de TGILS-SG() terminem suas execuções;
14     retorna s*;
15 fim
```

Algoritmo 2: PGILS-SG: pseudocódigo do algoritmo executado pela tarefa mestre.

4 Experimentos Computacionais

Este trabalho buscou primordialmente estudar o comportamento de um algoritmo heurístico sequencial eficiente quando em execução em um ambiente paralelo cooperativo. Uma vez que o algoritmo de [Silva *et al.* (2012)] apresenta atualmente o melhor desempenho, tanto em qualidade da solução quando em tempo de execução, apenas este trabalho foi considerado para efeitos comparativos.

Os experimentos foram realizados a partir de nove grupos de instâncias da literatura. Um conjunto, composto por 23 unidades e número de clientes variando entre 42 a 107, foi extraído do trabalho de [Abeledo *et al.* (2013)]. Outro grupo de 10 instâncias, contendo entre 70 a 532 clientes, foi selecionado da TSPLIB [Reinelt (1991)]. Finalmente, 9 conjuntos, cada um contendo 20 instâncias com 10, 20, 50, 100, 200, 500 e 1.000 clientes, foi obtido a partir de [Salehipour *et al.* (2011)].

O algoritmo desenvolvido foi codificado em linguagem C e compilado com o GNU gcc 4.6.3. Os experimentos foram executados em um equipamento com processador Intel® Core™ i7 2.93GHz, com 4 núcleos, 8MB de *cache* L3, 8GB de memória RAM, sob uma plataforma Linux Ubuntu 10.04 de 64 bits com *kernel* 2.6.32-25. As tarefas paralelas foram implementas via biblioteca POSIX *threads* (*pthreads*).

Os parâmetros de entrada do algoritmo foram definidos para $G_{max} = 10$ e $I_{max} = min\{100, n\}, R = \{0.00, 0.01, 0.02, \ldots, 0.25\}$, onde n representa o número de clientes da instância. O grau de paralelismo, ou número de tarefas paralelas, utilizado nos experimentos foi controlado pelo parâmetro de entrada p. Como medida de avaliação de desempenho do algoritmo paralelo foram utilizados o fator de *speedup* (1) e a eficiência (2) [Wilkinson e Allen (2005)]:

$$S(p) = \frac{\text{tempo de execução do algoritmo sequencial usando um único processador}}{\text{tempo de execução do algoritmo paralelo usando } p \text{ processadores}} \quad (1)$$

$$E(p) = \frac{S(p)}{p} \times 100\% \quad (2)$$

Como já mencionado, o algoritmo paralelo aqui proposto (PGILS-SG) foi desenvolvido tendo como referência o método sequencial de [Silva *et al.* (2012)]. Durante a implementação do algoritmo paralelo foram empregadas técnicas de otimização de código e estruturas de dados aprimoradas que contribuíram para a elaboração de uma aplicação eficiente. Este fato foi corroborado pelo desempenho apresentado por PGILS-SG quando executado sob as mesmas condições que a implementação sequencial original (GILS). Tal cenário pode ser obtido quando o algoritmo paralelo é executado com o parâmetro $p = 1$. Desta forma, uma única tarefa trabalhadora é lançada sob condições idênticas às experimentadas pelo algoritmo sequencial. Como não existem outras tarefas trabalhadoras colaborando, o mecanismo de compartilhamento de soluções não é acionado.

O resultado deste experimento pode ser visto na Tabela 1. Os valores médios reportados neste e nos demais ensaios deste trabalho são relativos a 10 execuções de cada instância e os parâmetros de entrada são os mesmos utilizados por GILS no trabalho original. As duas primeiras colunas da tabela representam, respectivamente, o grupo de instâncias selecionado e o trabalho de onde foram extraídas. A terceira coluna indica o número de clientes, enquanto a coluna seguinte mostra a quantidade de instâncias presentes em cada grupo. O tempo médio obtido por GILS para execução todas as instâncias de cada grupo é apresentado na quinta coluna. Na coluna seguinte, figura o resultado alcançado por PGILS-SG quando executado com apenas uma tarefa trabalhadora. Finalmente, a última coluna indica a taxa de melhoria do tempo de execução obtida por PGILS-SG neste experimento. Estas taxas foram aferidas pela razão entre os tempos médios de execução de GILS e PGILS-SG. As duas últimas linhas desta coluna apresentam a média e o coeficiente de variação (CV) das taxas de melhorias dos grupos de instâncias testados. Esta última aferição, determinada pela razão entre o desvio padrão e a média, é uma medida de diversidade que indica a homogeneidade de um conjunto dos dados, onde um valor não maior que 25% determina um conjunto com um bom grau de similaridade entre seus elementos [Triola (2013)]. Analisando os valores da última coluna da tabela, percebe-se que PGILS-SG foi mais em média rápido que GILS em todos os casos testados. A taxa de melhoria média final de $1,64$ e o coeficiente de variação de $14,60\%$ obtidos confirmam a eficiência das técnicas aplicadas na implementação de PGILS-SG. Esta comparação é justa porque todos os experimentos apresentados neste trabalho foram executados na mesma plataforma de *hardware* e *software* utilizada pelos autores do trabalho sequencial original.

Foram realizados experimentos com PGILS-SG executando múltiplas tarefas trabalhadoras visando avaliar o impacto do mecanismo de compartilhamento de soluções no desempenho do algoritmo. Nos resultados apresentados adiante, entenda-se por custo médio de uma solução, o valor da média entre os custos da melhor solução obtida em cada uma das 10 execuções do algoritmo. Já o termo *gap* refere-se à distância em pontos percentuais entre dois custos[1]. Em todos os casos, o algoritmo paralelo foi disparado com

[1]O *gap* de um valor v em relação a um valor de referência r é dado por: $gap(v, r) = 100 \times (v - r)/r$

Grupo	Fonte	Clientes	Qtde	Tempo (s)		Taxa de Melhoria
				GILS	PGILS-SG	
Grupo 1	Abeledo *et al.* (2013)	42 – 107	23	4,90	**2,51**	1,95
Grupo 2	Reinelt (1991)	70 – 532	10	273,38	**173,83**	1,57
Grupo 3	Salehipour *et al.* (2011)	10, 20, 50	60	0,19	**0,12**	1,58
Grupo 4	”	100	20	7,64	**4,75**	1,61
Grupo 5	”	200	20	72,08	**45,71**	1,58
Grupo 6	”	500	20	1.576,60	**1.269,32**	1,24
Grupo 7	”	1.000	20	28.186,14	**14.746,34**	1,91
		Total	**173**		**Média**	**1,64**
					CV %	**14,60**

Tabela 1: Comparação de desempenho entre GILS e PGILS-SG quando executado com apenas uma tarefa trabalhadora ($p = 1$).

4 tarefas trabalhadoras ($p = 4$), o mesmo número de núcleos de processamento disponíveis na máquina utilizada para execução dos experimentos.

A Tabela 2 apresenta os resultados obtidos por PGILS-SG para as instâncias dos Grupos de 1 a 5. As três primeiras colunas da tabela indicam, respectivamente, o nome, o número de clientes e a quantidade de instâncias presentes em cada grupo. Os tempos médios de execução obtidos pelo sequencial GILS podem ser vistos na quarta coluna. Cada linha da quinta coluna apresenta a média dos *gaps* entre os custos médios obtidos por PGILS-SG e a melhor solução encontrada pelo algoritmo sequencial. Um valor de *gap* negativo indica um desempenho superior do algoritmo paralelo. Os tempos médios de execução de PGILS-SG podem ser vistos na sexta coluna, enquanto o coeficiente de variação correspondente é apresentado na coluna seguinte. A última coluna exibe os valores de *speedup* obtidos por PGILS-SG em relação a GILS. Enfim, a última linha da tabela expõe a média dos valores apresentados nas respectivas colunas.

Grupo	Clientes	Qtde	GILS	PGILS-SG			
			Tempo Médio (s)	Custo	Tempo (s)		
				Gap %	Médio	CV %	Speedup
Grupo 1	42 - 107	23	4,90	0,0000	**0,91**	**15,82**	5,41
Grupo 2	70 - 532	10	273,38	−0,0104	**67,35**	**10,70**	4,06
Grupo 3	10, 20, 50	60	0,19	0,0000	**0,04**	**29,00**	4,52
Grupo 4	100	20	7,64	+0,0015	**1,74**	**13,69**	4,39
Grupo 5	200	20	72,08	+0,0111	**16,71**	**14,01**	4,31
		Média		**+0,0004**		**16,64**	**4,54**

Tabela 2: Resultados consolidados obtidos por PGILS-SG para as instâncias dos Grupos de 1 a 5 executando 4 tarefas trabalhadoras em uma máquina com 4 núcleos.

O algoritmo paralelo conseguiu alcançar a mesma solução que a implementação sequencial em todas as instâncias cujos dados estão consolidados na Tabela 2. A única exceção ocorreu com a instância att532 do Grupo 2, com 532 clientes, que teve o custo melhorado para um valor de 5.580.535 configurando, portanto, uma nova melhor solução. Os resultados relativos ao *gap* médio das soluções encontradas por PGILS-SG mostraram que o algoritmo apresentou um comportamento robusto, o que foi corroborado pelos irrisórios valores obtidos. Em relação ao tempo de execução, PGILS-SG apresentou *speedup* médio linear, quando $S(p) = p$, para todos os casos. Para os grupos de instâncias desta tabela, a eficiência média global foi igual 114% (Equação 2). Os valores do coeficiente de variação dos tempos médios de execução sugerem um desempenho estável, com pouca oscilação do tempo de execução.

Os resultados para as instâncias de maior porte, com 500 e 1.000 clientes, são destacados nas Tabelas 3 e 4. Nestas tabelas, a primeira coluna identifica a instância testada. A segunda coluna indica o melhor custo obtido por GILS, enquanto a terceira mostra o

gap do custo médio (GCM) em relação ao custo da melhor solução obtida pelo algoritmo sequencial. O tempo médio atingido por GILS pode ser visto na coluna 4. As demais colunas da tabela exibem resultados relativos a PGILS-SG. O custo da melhor solução é apresentado na quinta coluna, enquanto o *gap* entre o custo médio (GCM) e o custo da melhor solução pode ser visto na sexta coluna. A sétima coluna indica o *gap* entre os custos das melhores soluções de PGILS-SG em relação a GILS. Um valor negativo nesta coluna, assim como os valores em negrito correspondentes, indicam um melhor resultado percebido pelo algoritmo paralelo. As três últimas colunas apresentam os resultados de PGILS-SG relativos ao tempo de execução. A oitava coluna mostra o tempo médio de execução e a nona exibe o coeficiente de variação correspondente. Por fim, a última coluna relaciona os valores dos *speedups* médios de PGILS-SG em relação a GILS.

A Tabela 3 apresenta os resultados obtidos por PGILS-SG quando submetido às instâncias do Grupo 6, que possuem 500 clientes cada. Os valores destacados na coluna cinco mostram que PGILS-SG obteve 11 novas melhores soluções dentre as 20 instâncias deste grupo. A análise das terceira e sexta colunas, relativas aos *gaps* dos custos médios (GCM), mostra que o algoritmo paralelo apresenta um desempenho ligeiramente superior em termos de qualidade da solução que a implementação sequencial, como realçado pelos valores médios presentes na última linha da tabela.

Na Tabela 4, podem ser vistos os resultados referentes às instâncias do Grupo 7, que possuem 1.000 clientes cada. Os valores destacados na coluna cinco mostram que PGILS-SG obteve 13 novas melhores soluções para as instâncias deste grupo. Da mesma forma que para o grupo anterior, PGILS-SG mostra um comportamento estável, com pequenos valores de GCM, como pode ser visto na sexta coluna. O mesmo se repete quando se confrontam os custos da melhor solução obtida entre os dois algoritmos, que pode ser verificado pelos irrisórios valores de *gaps* projetados na sétima coluna.

Instância	GILS			PGILS-SG					
	Custo		Tempo Médio	Custo			Tempo		
	Melhor	GCM%		Melhor	GCM%	Gap%	Médio	CV%	*Speedup*
S500-R1	1.841.386	0,79	1.738,48	1.847.147	0,34	+0,31	383,19	13,76	4,54
S500-R2	1.816.568	0,36	1.476,13	1.817.453	0,21	+0,05	374,03	14.64	3,95
S500-R3	1.833.044	0,34	1.557,48	**1.829.192**	0,45	−0,21	362,24	21,33	4,30
S500-R4	1.809.266	0,37	1.597,06	**1.806.084**	0,35	−0,18	346,25	9,75	4,61
S500-R5	1.823.975	0,55	1.530,94	1.827.046	0,31	+0,17	326,29	8,72	4,69
S500-R6	1.786.620	0,24	1.576,91	1.787.148	0,24	+0,03	369,14	15,42	4,27
S500-R7	1.847.999	0,54	1.584,67	1.850.038	0,34	+0,11	373,65	18,08	4,24
S500-R8	1.820.846	0,46	1.565,01	1.827.098	0,23	+0,34	355,67	16,74	4,40
S500-R9	1.733.819	0,18	1.409,23	**1.729.400**	0,43	−0,25	340,45	14,42	4,14
S500-R10	1.762.741	0,26	1.621,85	**1.761.174**	0,37	−0,09	358,97	15,24	4,52
S500-R11	1.797.881	0,20	1.530,98	**1.796.680**	0,31	−0,07	345,35	21,40	4,43
S500-R12	1.774.452	0,53	1.554,75	1.777.297	0,23	+0,16	355,29	14,24	4,38
S500-R13	1.873.699	0,23	1.598,46	**1.868.910**	0,32	−0,26	366,82	16,43	4,36
S500-R14	1.799.171	0,36	1.701,90	1.804.296	0,17	+0,28	333,54	7,18	5,10
S500-R15	1.791.145	0,36	1.623,79	**1.785.263**	0,64	−0,33	379,62	19,29	4,28
S500-R16	1.810.188	0,35	1.583,70	**1.809.390**	0,31	−0,04	368,67	13,72	4,30
S500-R17	1.825.748	0,48	1.549,80	**1.824.673**	0,38	−0,06	378,90	10,57	4,09
S500-R18	1.826.263	0,27	1.620,02	**1.776.855**	0,46	−0,07	349,91	11,08	4,63
S500-R19	1.779.248	0,20	1.602,87	**1.776.855**	0,36	−0,13	321,32	16,38	4,99
S500-R20	1.820.813	0,53	1.507,96	1.821.773	0,51	+0,05	334,54	10,06	4,51
	Média	**0,39**			**0,34**			**14,41**	**4,44**

Tabela 3: Resultados obtidos por PGILS-SG para as instâncias do Grupo 6, com 500 clientes cada, executando 4 tarefas trabalhadoras em uma máquina com 4 núcleos.

A análise conjunta dos resultados presentes nas Tabelas 3 e 4 mostram que PGILS-SG possui um comportamento robusto demonstra a eficiência de PGILS-SG, que obteve soluções melhores ou muito próximas das atingidas pela versão sequencial. Este desempenho é interessante porque cada tarefa do algoritmo paralelo executa individualmente menos

Instância	GILS			PGILS-SG					
	Custo		Tempo Médio (s)	Custo			Tempo (s)		
	Melhor	GCM%		Melhor	GCM%	Gap%	Médio	CV%	*Speedup*
S1000-R1	5.107.395	0,52	31.894,51	5.108.146	0,43	+0,01	4.600,47	16,94	6,93
S1000-R2	5.106.161	0,42	30.881,19	5.109.610	0,39	+0,07	4.299,49	10,71	7,18
S1000-R3	5.096.977	0,32	30.184,15	**5.076.358**	0,51	−0,40	4.378,45	14,35	6,89
S1000-R4	5.118.006	0,46	29.951,12	**5.112.465**	0,58	−0,11	4.275,61	15,94	7,01
S1000-R5	5.103.894	0,37	30.129,51	**5.098.444**	0,64	−0,11	4.594,90	13,47	6,56
S1000-R6	5.115.816	0,53	28.161,57	5.121.932	0,50	+0,12	4.769,67	17,73	5,90
S1000-R7	5.021.383	0,23	25.945,41	**4.995.703**	0,78	−0,51	4.262,94	10,81	6,09
S1000-R8	5.109.325	0,46	26.572,71	5.125.120	0,27	+0,31	4.322,93	14,65	6,15
S1000-R9	5.052.599	0,41	26.330,40	**5.046.566**	0,56	−0,12	4.568,57	13,61	5,76
S1000-R10	5.078.191	0,30	25.676,31	**5.060.019**	0,81	−0,36	4.473,15	8,38	5,74
S1000-R11	5.041.913	0,48	26.235,63	**5.033.897**	0,58	−0,16	4.248,57	13,29	6,18
S1000-R12	5.029.792	0,43	27.910,11	5.031.234	0,37	+0,03	4.178,22	14,66	6,68
S1000-R13	5.102.520	0,57	28.475,89	**5.100.175**	0,59	−0,05	4.526,23	8,37	6,29
S1000-R14	5.099.433	0,38	27.639,81	**5.092.861**	0,48	−0,13	4.638,96	15,84	5,96
S1000-R15	5.142.470	0,62	27.633,07	**5.131.013**	0,75	−0,22	4.706,26	18,62	5,87
S1000-R16	5.073.972	0,32	26.653,16	**5.065.471**	0,38	−0,17	4.742,39	17,73	5,62
S1000-R17	5.071.485	0,26	27.503,43	**5.052.283**	0,66	−0,38	4.846,70	14,87	5,67
S1000-R18	5.017.589	0,39	28.808,09	5.019.823	0,52	+0,04	4.418,86	10,42	6,52
S1000-R19	5.076.800	0,40	29.637,49	**5.064.873**	0,53	−0,23	4.510,98	15,30	6,57
S1000-R20	4.977.262	0,52	27.499,24	5.001.616	0,26	+0,49	4.312,70	9,80	6,38
	Média	**0,42**			**0,53**			**13,77**	**6,30**

Tabela 4: Resultados obtidos por PGILS-SG para as instâncias do Grupo 7, com 1.000 clientes cada, executando 4 tarefas trabalhadoras em uma máquina com 4 núcleos.

iterações que a implementação sequencial, uma vez que estas são distribuídas entre as tarefas trabalhadoras. Desta forma, uma tarefa paralela que estava sendo conduzida em direção a uma solução de melhor qualidade pode, eventualmente, ser interrompida devido a expiração de suas iterações. Por outro lado, a despeito do número de iterações individual de cada tarefa, o mecanismo de compartilhamento de soluções compensa este desvio acelerando a convergência do algoritmo rumo a melhores resultados. Isto ocorre porque a busca local empenhada pelo algoritmo sequencial leva mais tempo para alcançar uma boa solução que estimule positivamente a convergência do algoritmo. Ambos os algoritmos, sequencial e paralelo, realizam buscas explorando sistematicamente diferentes regiões do espaço de soluções do problema. Enquanto o primeiro explora o espaço sequencialmente, uma região após a outra, o segundo realiza a busca simultaneamente em várias regiões, o que possibilita a obtenção de soluções de melhor custo mais rapidamente, induzindo a uma convergência mais acentuada.

Outro mérito do mecanismo de compartilhamento de soluções são anunciados pelos valores de *speedup* obtidos por PGILS-SG. Em princípio, devido à distribuição inicial de iterações, pode parecer que um valor de *speedup* linear seja o desempenho esperado para PGILS-SG. Contudo, as tarefas paralelas realizam trocas de soluções que, além do *overhead* intrínseco imposto pela comunicação, implicam em períodos de inatividade quando as tarefas concorrem pelo acesso a regiões críticas do programa.

Neste contexto, a análise dos resultados mostra que PGILS-SG apresentou desempenho superlinear, quando $S(p) > p$, para as instâncias de maior porte, como indicado pelos valores de *speedup* presentes na Tabela 4. De fato, para as instâncias do Grupo 7, o algoritmo alcançou eficiência média de 158%. Já para as instâncias da Tabela 3, PGILS-SG mostrou *speedup* médio linear, atingindo eficiência média igual a 111%, como pode ser observado na coluna correspondente. Este comportamento foi induzido pela composição dos seguintes fatores: a implementação eficiente do código paralelo, o mecanismo de compartilhamento de soluções e a natureza não determinística do algoritmo [Wilkinson e Allen (2005)].

Confrontando os valores de *speedup* presentes nas Tabelas 2, 3 e 4 percebe-se que PGILS-SG apresenta maior eficiência quando submetido a instâncias de maiores dimensões.

A razão para este comportamento está relacionada com o método de avaliação de movimentos e a forma como o algoritmo reage frente à frequência de atualização de sua solução corrente. Como abordado previamente, o método de avaliação calcula o custo de um movimento a partir da solução corrente em tempo constante. Para tanto, depende da construção de uma estrutura de dados auxiliar, a matriz de avaliação de movimentos, sempre que a solução corrente é atualizada. Isto significa que quanto maior a frequência de atualização da solução corrente, maior o tempo consumido no cômputo desta matriz. Uma vez que o algoritmo paralelo obtém soluções de qualidade mais rapidamente que a implementação sequencial, este tende a atualizar cada vez menos sua solução corrente e, consequentemente, deixa de despender um esforço significativo no cálculo da matriz de avaliação. Em instâncias de menor porte, este tempo não é tão relevante, face ao tempo total de execução, quanto para instâncias maiores. Porém, em instâncias de maior amplitude este tempo é expressivo, uma vez que o tamanho da matriz cresce quadraticamente em função da dimensão da instância.

5 Conclusão

Este trabalho introduziu uma heurística paralela híbrida eficiente para o Problema da Mínima Latência (PML). A abordagem utilizada empregou elementos de GRASP, ILS, RVND bem como uma técnica de avaliação de movimentos de tempo constante. A combinação destes elementos permitiram a concepção de um algoritmo paralelo cooperativo simples cuja eficácia foi comprovada por testes realizados em 173 instâncias. O algoritmo apresenta resultados bastante competitivos, tanto para a qualidade da solução quanto para o tempo de execução. De fato, 25 novas melhores soluções para o PML foram encontradas, enquanto o *speedup* obtido pelo algoritmo paralelo foi linear ou superlinear para as instâncias testadas. Tal desempenho deve-se, primordialmente, ao mecanismo de compartilhamento de soluções empenhado pelas tarefas paralelas, que acelera a convergência do algoritmo rumo a soluções de melhor qualidade, enquanto reduz significativamente o tempo de execução global.

Os resultados promissores obtidos dão margem a novas possibilidades de investigação. Experimentos preliminares ainda não publicados sugerem que um mecanismo de cooperação mais elaborado pode render resultados ainda melhores. Além disso, uma exploração na direção de outras arquiteturas paralelas ou do aumento do nível de paralelismo do algoritmo podem ser compensatórios.

6 Agradecimentos

Gostaríamos de agradecer aos autores do algoritmo sequencial GILS pela cessão dos equipamentos para execução dos experimentos e suporte prestado. Também dedicamos um especial agradecimento ao Prof. Luiz Satoru Ochi por suas relevantes contribuições para este artigo. Este trabalho foi parcialmente financiado pelo CNPq, FAPERJ e UESPI.

Referências

Abeledo, H., Fukasawa, R., Pessoa, A. e Uchoa, E. (2013), The time dependent traveling salesman problem: polyhedra and algorithm. *Math. Program. Comput.*, v. 5, n. 1, p. 27–55.

Angel-Bello, F., Alvarez, A. e García, I. (2013), Two improved formulations for the minimum latency problem. *Applied Math. Modelling*, v. 37, n. 4, p. 2257 – 2266.

Araújo, A. P., Boeres, C., Rebello, V. E., Ribeiro, C. C. e Urrutia, S. Exploring grid implementations of parallel cooperative metaheuristics. *Metaheuristics*, p. 297–322. Springer, 2007.

Arora, S. e Karakostas, G. (2003), Approximation schemes for minimum latency problems. *SIAM J. on Computing*, v. 32, n. 5, p. 1317–1337.

Blum, A., Chalasani, P., Coppersmith, D., Pulleyblank, B., Raghavan, P. e Sudan, M. The minimum latency problem. *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, p. 163–171. ACM, 1994.

Chen, P., Dong, X. e Niu, Y. An iterated local search algorithm for the cumulative capacitated vehicle routing problem. Tan, H. (Ed.), *Technology for Education and Learning*, v. 136 of *Advances in Intel. and Soft Computing*, p. 575–581. ICTEL, 2012.

Coelho, I. M., Haddad, M. N., Ochi, L. S., Souza, M. J. F. e Farias, R. A hybrid cpu-gpu local search heuristic for the unrelated parallel machine scheduling problem. *3rd Workshop on Applications for Multi-Core Architectures (WAMCA), 2012*, p. 19–23. IEEE, 2012.

Dewilde, T., Cattrysse, D., Coene, S., Spieksma, F. C. e Vansteenwegen, P. (2013), Heuristics for the traveling repairman problem with profits. *C&OR*.

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

Feo, T. e Resende, M. (1995), Greedy randomized adaptive search procedures. *J. of Global Optimizartion*, v. 6, n. 2, p. 109–133.

Gendreau, M. e Potvin, J.-Y. *Handbook of Metaheuristics*. Springer, 2010.

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

Reinelt, G. (1991), Tsplib: A traveling salesman problem library. *ORSA J. on Computing*, v. 3, n. 4, p. 376–384.

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

Salehipour, A., Sorensen, K., Goos, P. e Braysy, O. (2011), Efficient GRASP+VND and GRASP+VNS metaheuristics for the traveling repairman problem. *4OR-A Quarterly J. of Oper. Res.*, v. 9, n. 2, p. 189–209.

Silva, M. M., Subramanian, A., Vidal, T. e Ochi, L. S. (2012), A simple and effective metaheuristic for the minimum latency problem. *EJOR*, v. 221, n. 3, p. 513 – 520.

Sitters, R. *The minimum latency problem is NP-hard for weighted trees*. Springer, 2006.

Subramanian, A., Drummond, L., Bentes, C., Ochi, L. e Farias, R. (2010), A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, v. 37, n. 11, p. 1899–1911.

Triola, M. F. *Introdução à Estatística: atualização da tecnologia*. LTC, 11 edição, 2013.

Wilkinson, B. e Allen, M. *Parallel Programming: techniques and applications using networked workstations and parallel computers*. Prentice Hall, 2a. edição, 2005.