Iterated local search para problemas de ordenação linear de grande porte

Celso S. Sakuraba[a], Débora P. Ronconi[a] e Mutsunori Yagiura[b]

[a]*Departamento de Engenharia de Produção, Escola Politécnica, Universidade de São Paulo*
Av. Prof. Almeida Prado, Trav. 2, Nº 128 - Cidade Universitária, São Paulo 05508-070 Brazil
sakuraba@usp.br,dronconi@usp.br
[b]*Department of Computer Science and Mathematical Informatics, Graduate School of Information Science, Nagoya University, Furocho, Chikusaku, Nagoya 464-8603, Japan*
yagiura@nagoya-u.jp

Resumo. Dado um grafo com n vértices, m arcos e custos nos arcos, o problema de ordenação linear consiste em encontrar uma permutação π dos vértices de forma a minimizar o custo dos arcos reversos. Sua aplicação mais conhecida consiste na triangulação de matrizes de entradas e saídas, que possibilita análises econômicas em relação à estabilidade de uma determinda região. As diversas metaheurísticas existentes para este problema se utilizam de busca local com a vizinhança de inserção, que se implementada sem a utilização de estruturas de dados eficientes, se torna inviável para problemas de maiores dimensões, como por exemplo grafos com milhares de vértices. Apresentamos uma implementação da metaheurísitca *iterated local search* que utiliza um algoritmo de busca local eficiente, e pode ser equiparada a outras metaheurísticas para problemas menores, além de obter um novo *benchmark* de valores de soluções para instâncias com 500 ou mais vértices.
Palavras chave: problema de ordenação linear, metaheurísticas, *iterated local search*.

Abstract. Given a directed graph with n vertices, m edges and costs on the edges, the linear ordering problem consists of finding a permutation π of the vertices so that the total cost of the reverse edges is minimized. Its most widely known application is the triangularization of input-output matrices, which allows the analysis of the economical stability of a certain region. Various metaheuristics developed to approach this problem use local search with the insert neighborhood. This local search, if implemented without the usage of an efficient algorithm, cannot handle large instances, as for example graphs with thousands of vertices. We present an implementation of iterated local search that uses an efficient local search algorithm, whose performance can be compared with other metaheuristics for small instances and can obtain a new benckmark of solutions for instances with 500 vertices or more.
Keywords: linear ordering problem, metaheuristics, iterated local search.

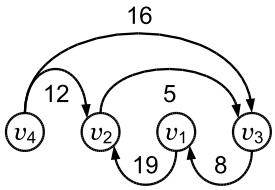
1 Introdução

Dado um grafo orientado $G = (V, E)$ com um conjunto de vértices V ($|V| = n$), um conjunto de arcos $E \subseteq V \times V$ e um custo c_{uv} para cada arco (u, v), o problema de ordenação linear (POL) consiste em encontrar uma permutação de vértices que minimize o custo total dos arcos reversos, i.e., arcos saindo de um vértice u e entrando em um vértice v, sendo que v está em uma posição anterior a u na permutação. É assumido sem perda de generalidade que $c_{uv} > 0$ é válido para todo $(u, v) \in E$ e que se tomarmos G como

um grafo não-orientado, G é conectado (o que implica $m \geq n-1$, onde $m = |E|$). Por conveniência, assumimos também $c_{uv} = 0$ para todo $(u, v) \notin E$. Denotando a permutação por $\pi : \{1, \ldots, n\} \rightarrow V$, onde $\pi(i) = v$ significa que v é o $i$ésimo elemento de π, o custo total dos arcos reversos é formalmente definido por:

$$custo(\pi) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{\pi(j)\pi(i)}. \quad (1)$$

Outra representação do POL consiste em encontrar uma permutação de n índices de forma que quando permutamos as linhas e colunas de uma matriz $n \times n$ com esta permutação (note que a mesma permutação é utilizada para linhas e colunas), a soma dos valores no triângulo superior (i.e., valores acima da diagonal) é maximizada. A equivalência entre as representações é imediata, e.g., considerando a matriz de adjacência de G, como pode ser visto no exemplo da Figura 1. Na figura, cada vértice v_i corresponde a uma linha e a uma coluna da matriz.



	4	2	1	3
4	0	12	0	16
2	0	0	0	5
1	0	19	0	0
3	0	0	8	0

Figura 1: Representações em forma de grafo e matriz de uma solução do POL com custo 19+8=27

Em sua representação em forma de matriz, o POL possui a seguinte formulação:

$$\begin{array}{lll} \text{maximize} & \sum_{i \in V} \sum_{\substack{j \in V \\ j \neq i}} c_{ij} x_{ij} & \\ \text{sujeito a} & x_{ij} + x_{ji} = 1 & \forall i, j \in V, i \neq j \\ & x_{ij} + x_{jk} + x_{ki} \leq 2 & \forall i, j, k \in V, i \neq j \neq k \neq i \\ & x_{ij} \in \{0, 1\} & \forall i, j \in V. \end{array}$$

O POL possui aplicações em diversas áreas (Grötschel *et al.*, 1984), dentre as quais a mais conhecida é a triangularização de matrizes de entradas e saídas (*input-output matrices*), que permite a economistas analizar a estabilidade econômica de uma dada região. O POL é NP-difícil (Karp, 1972; Garey e Johnson, 1979), e tem sido extensivamente estudado desde a publicação do primeiro artigo a seu respeito (Chenery e Watanabe, 1958), tendo sido desenvolvidos diversos algoritmos exatos e heurísticos para resolvê-lo. Revisões de literatura sobre o POL são apresentadas por Schiavinotto e Stützle (2004) e Charon e Hudry (2007).

Dentre as abordagens heurísticas, há diversas metaheurísticas para lidar com o POL, tais como busca tabu (Laguna *et al.*, 1999), *scatter search* (Campos *et al.*, 2001), algoritmo genético (Huang e Lim, 2003) e *variable neighborhood search* (Garcia *et al.*, 2006). Tais metaheurísticas fazem uso de busca local intrinsecamente para obter soluções de boa qualidade. As vizinhanças de busca mais conhecidas para o POL são aquelas dadas pelas seguintes operações:

- *inserção*: remova o vértice de uma posição i e insira-o depois (resp., antes) do vértice na posição j para $i < j$ (resp., $i > j$);
- *troca*: troque os vértices nas posições i e j.

Huang e Lim (2003) demonstraram que apesar de uma solução que pode ser melhorada por uma troca poder sempre ser melhorada por uma inserção, nem toda solução que pode ser melhorada por uma inserção pode ser melhorada por uma troca. Como esperado, experimentos conduzidos por Schiavinotto e Stützle (2004) mostram que a vizinhança de inserção possui melhores resultados do que a de troca, e todas as metaheurísticas citadas acima fazem uso da vizinhança de inserção.

Seja π' a permutação obtida a partir de uma permutação π após a inserção do vértice da posição i na posição j. A diferença entre os custos dessas permutações é dada por:

$$custo(\pi') - custo(\pi) = \begin{cases} \sum_{k=i+1}^{j} \left(c_{\pi(i)\pi(k)} - c_{\pi(k)\pi(i)}\right), & i < j \\ \sum_{k=j}^{i-1} \left(c_{\pi(k)\pi(i)} - c_{\pi(i)\pi(k)}\right), & i > j. \end{cases} \quad (2)$$

A diferença de custo ocasionada por uma inserção pode ser calculada em tempo $O(n)$, o que faz com que uma simples busca através da vizinhança de inserção seja possível em tempo $O(n^3)$. Este tempo pode ser reduzido para $O(n^2)$ se conduzirmos a busca de forma ordenada, conforme descrito em Schiavinotto e Stützle (2004).

Sakuraba e Yagiura (2010) demonstraram que algoritmos que utilizam busca local com tempo $O(n^2)$ repetidamente se tornam inviáveis quando lidamos com instâncias grandes, e.g., instâncias com milhares de vértices. Estes autores propõem um algoritmo, denominado TREE, que realiza a busca local na vizinhança de inserção em tempo $O(n+\Delta \log \Delta)$, onde Δ representa o máximo grau do grafo.

Neste trabalho, buscamos explorar as propriedades do algoritmo TREE, verificando sua eficácia quando inserido na estrutura de uma metaherística, no caso a *iterated local search*. Desenvolvemos uma metaheurística capaz de encontrar soluções de boa qualidade para instâncias grandes, além de encontrar soluções ótimas para instâncias para as quais estas soluções são conhecidas. A seção seguinte apresenta o conceio da metaheurística *iterated local search*, além de descrever o algoritmo desenvolvido para o POL. As seções subsequentes mostram os resultados obtidos em experimentos computacionais e as conclusões deste trabalho.

2 Descrição do Algoritmo

A metaheurística desenvolvida neste trabalho utiliza o algoritmo de busca local TREE, que segundo experimentos apresentados em Sakuraba e Yagiura (2010), é o único algoritmo capaz de encontrar ótimos locais para instâncias grandes em um tempo razoável.

O algoritmo TREE utiliza árvores de busca balanceadas para realizar a busca pela vizinhança de inserção do POL de maneira eficiente. Uma árvore cujos nós internos possuem sempre dois ou três filhos é construída para cada vértice, e a árvore correspondente ao vértice v é utilizada para calcular o custo das soluções obtidas através da inserção deste vértice em diferentes posições de π. Um exemplo de árvore é dado na Figura 2.

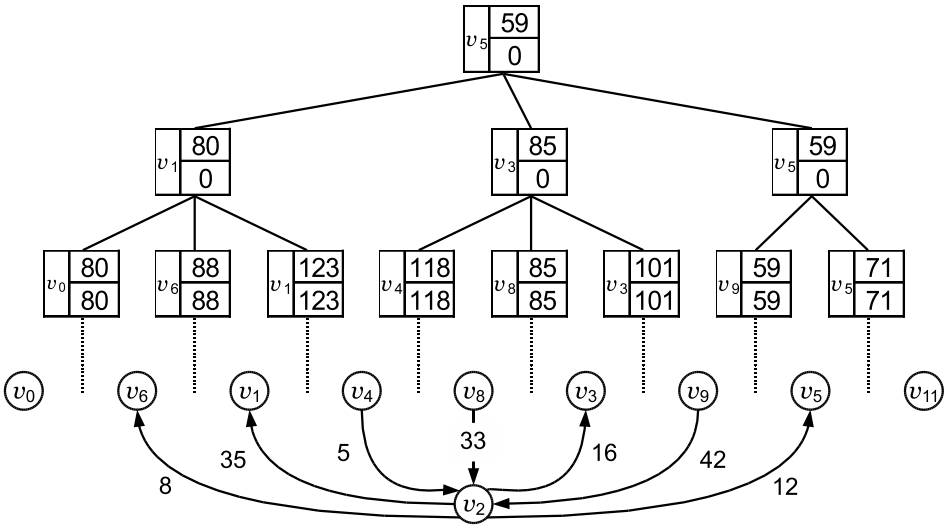


Figura 2: Os vértices adjacentes e a árvore correspondente a um vértice v_2.

A parte inferior da figura representa os vértices adjacentes a v_2 e as direções e custos dos arcos que os conectam. A parte superior apresenta a árvore correspondente a v_2, onde o valor presente na raiz nos permite calcular o ganho máximo ao inserir v_2 em uma determinada posição. Os valores nos nós internos auxiliam o cálculo do valor na raiz e possibilitam o cálculo do custo de inserções em diferentes posições. Através da manutenção dos valores corretos nos nós das árvores, é possível realizar as seguintes operações:

- Encontrar um vértice v, que ao ser inserido em uma determinada posição gere uma solução π' de menor custo do que a solução atual π, ou concluir que tal vértice não existe, em tempo $O(n)$;
- Encontrar a posição onde v deve ser inserido para gerar π' em tempo $O(\log d_v)$, onde d_v é o grau de v;
- Atualizar as árvores de acordo com a nova solução, em $O(d_v \log \Delta)$.

Veja Sakuraba e Yagiura (2010) para uma descrição detalhada do funcionamento do algoritmo TREE. Este algoritmo realiza a busca local pela vizinhança de inserção em tempo $O(n + \Delta \log \Delta)$, podendo ser utilizado dentro de metaheurísticas gerando soluções de qualidade superior. Apesar de sua eficiência para realizar a busca local, o algoritmo TREE consome uma quantidade razoável de esforço computacional na geração inicial de sua estrutura de dados, além de uma quantidade considerável de memória para mantê-la. Tais características adequam-se à inserção do algoritmo TREE em metaheurísticas de trajetória, tais como a *iterated local search*.

Iterated local search (ILS) é uma metaheurística introduzida por Baxter (1981), e que pode ser descrita resumidamente em três passos. Primeiramente, gere uma solução e aplique uma busca local de forma a obter uma solução localmente ótima s^*. Em seguida, aplique uma perturbação e uma busca local, obtendo $s^{*\prime}$. Repita o passo anterior até que uma condição de término seja satisfeita, utilizando $s^{*\prime}$ caso $s^{*\prime}$ satisfaça um critério de aceitação, ou s^* caso contrário. No caso de perturbações determinísticas, pode-se utilizar

uma memória para evitar ciclos curtos (Lourenço *et al.*, 2003). O Algoritmo 1 descreve a estrutura básica da *iterated local search*.

Algoritmo 1 Estrutura da *iterated local search*

s_0 = *GerarSoluçãoInicial*
s^* = *BuscaLocal*(s_0)
while Condição de término não é satisfeita **do**
  s' = *Perturbação*(s^*, *histórico*)
  $s^{*\prime}$ = *BuscaLocal*(s')
  s^* = *CritérioAceitação*(s^*, $s^{*\prime}$, *histórico*)
end while

Analisando a estrutura da *iterated local search*, vemos que além de determinar o algoritmo utilizado na busca local (no caso, o algoritmo TREE), é preciso definir pelo mais três funções: a solução inicial, a perturbação e o critério de aceitação.

Para gerar a solução inicial, utilizamos uma variação da heurística construtiva proposta por Becker (1967). A heurística original utiliza a representação em forma de matriz do POL para calcular quocientes para cada elemento (que equivale a um vértice do grafo), dados pela divisão da soma dos valores na linha pela soma dos valores na coluna correspondente aquele elemento. A solução é obtida classificando os elementos em ordem crescente do valor dos quocientes. A variação utilizada neste trabalho realiza um cálculo semelhante, mas substituindo a soma dos valores na linha pela soma dos arcos que saem, e a soma dos valores na coluna pela soma dos arcos que entram em um vértice.

Como perturbação, foram utilizados k movimentos de inserção escolhendo vértices v aleatoriamente através de uma distribuição de probabilidade uniforme. Para determinar a posição em que v é inserido, percorre-se a árvore de v da raiz até uma folha observando o valor de $\gamma_{\min}(x)$, representado na parte superior de cada nó x na árvore da Figura 2. $\gamma_{\min}(x)$ é mantido de tal forma que ao percorrermos a árvore da raiz até uma folha escolhendo sempre o nó filho com o menor valor de $\gamma_{\min}(x)$, chegaremos à folha equivalente à posição onde v seria inserido para gerar uma solução de menor custo. Em uma perturbação, percorremos a árvore através do nó filho que possua o maior valor de $\gamma_{\min}(x)$, de forma a inserir v em posições diferentes daquelas normalmente inseridas pela busca local. (Veja Sakuraba e Yagiura (2010) para uma descrição detalhada de $\gamma_{\min}(x)$ e da busca local utilizando o algoritmo TREE.)

Como critério de aceitação, escolhemos sempre aceitar a solução localmente ótima atual, dado o esforço computacional necessário para reconstruir ou armazenar a estrutura de dados correspondente à solução anterior (ou seja, sempre aceita-se a nova solução e parte-se dela para a próxima etapa).

3 Resultados Computacionais

Para avaliar a metaheurística ILS, comparamos seu desempenho utilizando os valores das melhores soluções encontradas até o momento para diversos tipos de instâncias, divididas em três grupos de acordo com seus tamanhos:

- **pequenas**: matrizes com até 75 setores contendo dados de economias européias e americana. São conhecidas soluções ótimas para todas as instâncias deste grupo.

LOLIB: 49 instâncias relativas a matrizes de entradas e saídas (*input-output matrices*) da economia européia. Elas possuem entre 44 e 60 setores.

SGB: 25 instâncias relativas à matrizes de entradas e saídas de 75 setores da economia norte-americana.

- **médias**: matrizes com até 200 setores, para as quais as soluções ótmas não são conhecidas.

  Random I: 75 instâncias com 100, 150 ou 200 vértices geradas através de uma distribuição uniforme.

  Random II: 75 instâncias com 100, 150 ou 200 vértices geradas pela contagem do número de vezes que um setor aparece depois de outro em um conjunto de permutações geradas aleatoriamente.

- **grandes**: 150 instâncias com tamanhos entre 500 e 8000 vértices geradas utilizando uma distribuição uniforme e densidades (probabilidade de existir um arco entre dois vértices) controladas. Para estas instâncias, é necessário utilizar o algoritmo TREE para se obter soluções localmente ótimas em tempo razoável e não foram encontrados resultados obtidos por outras metaheurísticas na literatura.

As instâncias descritas acima podem ser obtidas em http://heur.uv.es/optsicom/LOLIB/ e http://www.hirata.nuee.nagoya-u.ac.jp/~yagiura/lop/. Os valores de soluções ótimas ou das melhores soluções conhecidas podem ser encontrados em Garcia *et al.* (2006) (instâncias pequenas e médias) e Sakuraba e Yagiura (2010) (instâncias grandes).

A metaheurística ILS proposta foi implementada em linguagem C, e os experimentos foram realizados em um Intel Xeon 2.83 GHz com 24GB de RAM. Os resultados são apresentados na Tabela 1. Os valores das soluções foram calculados em termos da soma dos elemenos acima da diagonal (representação em forma de matriz) para permitir uma comparação com outros resultados da literatura.

A primeira coluna da tabela descreve as instâncias utilizadas. As instâncias do tipo Random I, Random II e grandes são identificadas respectivamente da seguinte maneira: t1d[*número de vértices*], t2d[*número de vértices*] e n[*número de vértices*]d[*densidade*]. Os pares de colunas seguintes descrevem o número de soluções com valores melhores (M), iguais (I) ou piores (P) encontradas pela metaheurística proposta em relação às soluções ótimas ou às melhores soluções reportadas na literatura, além do desvio relativo percentual entre o valor da soma dos elementos no triângulo superior destas soluções ($trisup_{\text{opt/best}}$) e aquele encontrado pela metaheurística ILS ($trisup_{\text{ILS}}$), calculado pela fórmula $100 * (trisup_{\text{ILS}} - trisup_{\text{opt/best}})/trisup_{\text{opt/best}}$. Esses resultados são divididos segundo o número de perturbações k realizadas em cada etapa de perturbação do algoritmo e o limite de tempo em segundos. Os limites de tempo foram estabelecidos de forma a serem compatíveis com os tempos reportados por outras metaheurísticas para o POL citadas na Seção 1.

Podemos observar na Tabela 1 que as soluções ótimas para instâncias pequenas foram alcançadas em quase todos os testes. Foram obtidos resultados melhores do que os encontrados na literatura para a maior parte das instâncias do tipo Random I com $k = n/2$ e $n/7$ segundos. Para as instâncias do tipo Random II, apenas três soluções melhoradas foram encontradas. Porém a média dos desvios dos valores das soluções obtidas para estas instâncias é no máximo -0,002%.

Tabela 1: Comparação com os melhores resultados encontrados na literatura

instância	$k = n/2$ perturbações				$k = n/5$ perturbações			
	tempo = $n/10$ s		tempo = $n/7$ s		tempo = $n/10$ s		tempo = $n/7$ s	
	M/I/P	des%	M/I/P	des%	M/I/P	des%	M/I/P	des%
LOLIB	0/49/0	0	0/49/0	0	0/48/1	≈ 0	0/48/1	≈ 0
SGB	0/24/1	≈ 0	0/24/1	≈ 0	0/23/2	≈ 0	0/24/1	≈ 0
t1d100	12/2/11	0,0059	12/4/9	0,0095	6/3/16	-0,0357	6/3/16	-0,0285
t1d150	15/0/10	0,0089	19/0/6	0,0173	4/0/21	-0,0442	6/0/19	-0,0333
t1d200	17/0/8	0,0259	21/0/4	0,0346	3/0/22	-0,0692	3/0/22	-0,0562
t2d100	0/12/13	-0,0011	0/15/10	-0,0008	0/21/4	-0,0002	0/23/2	-0,0001
t2d150	0/1/24	-0,0014	0/1/24	-0,0013	0/4/21	-0,0007	0/7/18	-0,0004
t2d200	1/0/24	-0,0020	1/0/24	-0,0018	3/1/21	-0,0009	3/2/20	-0,0006
n0500d001	5/0/0	2,7578	5/0/0	2,7760	5/0/0	3,4930	5/0/0	3,5646
n0500d005	5/0/0	2,3226	5/0/0	2,3627	5/0/0	2,9338	5/0/0	2,9493
n0500d010	5/0/0	2,0391	5/0/0	2,0618	5/0/0	2,2726	5/0/0	2,3158
n0500d050	5/0/0	0,6533	5/0/0	0,6810	5/0/0	0,4554	5/0/0	0,4913
n0500d100	5/0/0	0,2066	5/0/0	0,2190	5/0/0	0,1476	5/0/0	0,1570
n1000d001	5/0/0	2,4168	5/0/0	2,4312	5/0/0	3,1857	5/0/0	3,1866
n1000d005	5/0/0	2,0705	5/0/0	2,0942	5/0/0	2,2488	5/0/0	2,2983
n1000d010	5/0/0	1,4429	5/0/0	1,4795	5/0/0	1,4004	5/0/0	1,4404
n1000d050	5/0/0	0,4524	5/0/0	0,4787	5/0/0	0,2462	5/0/0	0,2638
n1000d100	5/0/0	0,1101	5/0/0	0,1218	5/0/0	0,0352	5/0/0	0,0424
n2000d001	5/0/0	2,2914	5/0/0	2,3058	5/0/0	2,9292	5/0/0	2,9831
n2000d005	5/0/0	1,6087	5/0/0	1,6338	5/0/0	1,4839	5/0/0	1,5203
n2000d010	5/0/0	1,0545	5/0/0	1,0883	5/0/0	0,7829	5/0/0	0,8456
n2000d050	5/0/0	0,2211	5/0/0	0,2476	5/0/0	0,0718	5/0/0	0,0913
n2000d100	5/0/0	0,0401	5/0/0	0,0499	3/0/2	0,0022	3/0/2	0,0071
n3000d001	5/0/0	2,1249	5/0/0	2,1330	5/0/0	2,6483	5/0/0	2,6762
n3000d005	5/0/0	1,2203	5/0/0	1,2706	5/0/0	1,0426	5/0/0	1,0998
n3000d010	5/0/0	0,7470	5/0/0	0,7853	5/0/0	0,5321	5/0/0	0,5832
n3000d050	5/0/0	0,1359	5/0/0	0,1647	5/0/0	0,0552	5/0/0	0,0696
n3000d100	5/0/0	0,0269	5/0/0	0,0368	4/0/1	0,0089	5/0/0	0,0125
n4000d001	5/0/0	2,0844	5/0/0	2,1152	5/0/0	2,4398	5/0/0	2,4903
n4000d005	5/0/0	1,0621	5/0/0	1,1160	5/0/0	0,8164	5/0/0	0,8753
n4000d010	5/0/0	0,6101	5/0/0	0,6589	5/0/0	0,3945	5/0/0	0,4262
n4000d050	5/0/0	0,1093	5/0/0	0,1342	5/0/0	0,0369	5/0/0	0,0477
n4000d100	5/0/0	0,0103	5/0/0	0,0167	3/0/2	0,0000	4/0/1	0,0036
n8000d001	5/0/0	1,7097	5/0/0	1,7525	5/0/0	1,6633	5/0/0	1,7451
n8000d005	5/0/0	0,6202	5/0/0	0,6617	5/0/0	0,4061	5/0/0	0,4424
n8000d010	5/0/0	0,3251	5/0/0	0,3584	5/0/0	0,1599	5/0/0	0,1868
n8000d050	5/0/0	0,0287	5/0/0	0,0427	4/0/1	0,0033	4/0/1	0,0091
n8000d100	2/0/3	-0,0008	5/0/0	0,0075	2/0/3	-0,0008	3/0/2	0,0019

Melhores soluções foram encontradas para as 150 instâncias grandes, com melhorias de até 4,44% em relação aos melhores resultados disponívies na literatura, obtidos pelo algoritmo de busca local TREE. Para alguns tipos de instâncias, foi possível observar

uma melhoria média de aproximadamente 3,5% em um tempo menor do que um minuto. Observando os resultados obtidos com estas instâncias, podemos ver uma clara influência da densidade das amostras e do número de perturbações. Para as instâncias de tamanhos entre 500 e 4000 vértices e densidade 1%, observamos que os testes feitos com $k = n/5$ obtiveram melhores resultados do que aqueles com $k = n/2$. O mesmo ocorreu para as instâncias com $n = 500$ e densidades 5 e 10%, e $n = 1000$ e densidade 5%, sugerindo que instâncias menores e mais esparsas obtenham melhores resultados com valores menores de k.

Como último experimento, testou-se a performance do algoritmo proposto utilizando um limite de tempo maior. Foram escolhidos como parâmetros $k = n/2$ e um limite de tempo computacional de n segundos. Os resultados são mostrados na Tabela 2.

Tabela 2: Resultados obtidos com limite de tempo de n segundos

	M/I/P	des%	M/I/P	des%	M/I/P	des%
	LOLIB		SGB			
	0/49/0	0	0/25/0	0		
	Random I					
	$n = 100$		$n = 150$		$n = 200$	
	15/6/4	0,0144	22/0/3	0,0349	25/0/0	0,0569
	Random II					
	$n = 100$		$n = 150$		$n = 200$	
	0/23/2	-0,0001	0/2/23	-0,0007	1/1/23	-0,0010
dens.	grandes					
	$n0500$		$n1000$		$n2000$	
1%	5/0/0	2,9011	5/0/0	2,5648	5/0/0	2,3281
5%	5/0/0	2,4611	5/0/0	2,1721	5/0/0	1,7678
10%	5/0/0	2,1651	5/0/0	1,6050	5/0/0	1,2512
50%	5/0/0	0,7867	5/0/0	0,5753	5/0/0	0,3517
100%	5/0/0	0,2683	5/0/0	0,1645	5/0/0	0,0894
	$n3000$		$n4000$		$n8000$	
1%	5/0/0	2,1636	5/0/0	2,1480	5/0/0	1,8681
5%	5/0/0	1,4429	5/0/0	1,2700	5/0/0	0,8203
10%	5/0/0	0,9412	5/0/0	0,8153	5/0/0	0,4893
50%	5/0/0	0,2562	5/0/0	0,2146	5/0/0	0,1134
100%	5/0/0	0,0680	5/0/0	0,0496	5/0/0	0,0280

Os valores são calculados e apresentados da mesma forma que a Tabela 1. Podemos notar que com um limite de tempo maior, pudemos obter soluções ótimas para todas as instâncias menores, além de soluções melhores do que as obtidas nos experimentos anteriores para diversas instâncias.

As tabelas presentes no Anexo I mostram os novos valores de todas as soluções que foram melhoradas em alguns dos experimentos e que podem ser usadas para futuras comparações.

4 Conclusões e Trabalhos Futuros

Neste trabalho foi verificada a eficácia da utilização do algoritmo TREE, que realiza a busca local pela vizinhança de inserção do problema de ordenação linear de maneira eficaz, em conjunto com a metaheurística *iterated local search* visando o desenvolvimento de técnicas capazes de lidar com instâncias com um grande número de vértices.

Através do desenvolvimento de uma metaheurística *iterated local search* que explora os pontos fortes do algoritmo TREE, obteve-se bons resultados sem a necessidade da calibração de muitos parâmetros. Tais resultados incluem a obtenção de soluções ótimas para instâncias pequenas e soluções melhoradas para instâncias com até 200 vértices. Além disso, foi criado um novo *benchmark* de soluções para instâncias maiores, com até 8000 vértices, para os quais estavam disponíveis apenas soluções obtidas através da utilização de busca local.

Como próximos passos pretende-se refinar a metaheurística desenvolvida de forma a torná-la mais eficiente, e compará-la com outras metaheurísticas que também possam lidar com instâncias de grande porte.

Agradecimentos

Os autores gostariam de agradecer os auxílios financeiros recebidos do Conselho Nacional de Desenvolvimento Científico e Tecnológico e da Fundação de Amparo à Pesquisa do Estado de São Paulo.

Referências

Baxter, J.: 1981, Local optima avoidance in depot location, *The Journal of the Operational Research Society* **32**(9), 815–819.

Becker, O.: 1967, Das helmstädtersche reihenfolgeproblem - die effizienz verschiedener näherungsverfahren, *Computer uses in the Social Science* .

Campos, V., Glover, F., Laguna, M. e Martí, R.: 2001, An experimental evaluation of a scatter search for the linear ordering problem, *Journal of Global Optimization* **21**(4), 397–414.

Charon, I. e Hudry, O.: 2007, A survey on the linear ordering problem for weighted or unweighted tournaments, *4OR: A Quarterly Journal of Operations Research* **5**, 5–60.

Chenery, H. B. e Watanabe, T.: 1958, International comparisons of the structure of production, *Econometrica* **26**, 487–521.

Garcia, C. G., Pérez-Brito, D., Campos, V. e Martí, R.: 2006, Variable neighborhood search for the linear ordering problem, *Computers and Operations Research* **33**, 3549–3565.

Garey, M. R. e Johnson, D. S.: 1979, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co Ltd, New York, NY, USA.

Grötschel, M., Jünger, M. e Reinelt, G.: 1984, A cutting plane algorithm for the linear ordering problem, *Operations Research* **32**, 1195–1220.

Huang, G. e Lim, A.: 2003, Designing a hybrid genetic algorithm for the linear ordering problem, *Genetic and Evolutionary Computation - GECCO 2003, proc.* pp. 1053–1064.

Karp, R. M.: 1972, Reducibility among combinatorial problems, *in* R. E. Miller e J. W. Thatcher (eds), *Complexity of Computer Computations*, Plenum Press, New York, NY, USA, pp. 85–103.

Laguna, M., Marti, R. e Campos, V.: 1999, Intensification and diversification with elite

tabu search solutions for the linear ordering problem, *Computers and Operations Research* **26**, 1217–1230.

Lourenço, H. R., Martin, O. C. e Stützle, T.: 2003, Iterated local search, *in* F. Glover e G. A. Kochenberger (eds), *Handbook of Metaheuristics*, Kluwer Academic Publishers, chapter 11, pp. 321–353.

Sakuraba, C. S. e Yagiura, M.: 2010, Efficient local search algorithms for the linear ordering problem, *International Transactions in Operational Research* **17**(6), 711–737.

Schiavinotto, T. e Stützle, T.: 2004, The linear ordering problem: Instances, search space analysis and algorithms, *Journal of Mathematical Modelling and Algorithms* **3**, 367–402.

Anexo I

As Tabelas 3, 4 e 5 a seguir apresentam valores de soluções encontradas por nossa *iterated local search* que superaram os melhores valores encontrados na literatura. Os valores são dados em função da soma dos valores no triângulo superior para o problema de maximização, e foram obtidos em experimentos com $k = n/2$ perturbações e limite de tempo de n segundos.

A Tabela 3 apresenta as instâncias divididas de acordo com o tipo (Random I ou Random II) e número de setores, sendo cada instância identificada por [*número de setores*].[*número da instância*]. As Tabelas 4 e 5 estão divididas de acordo com o número de vértices e identificadas por d[*densidade*]-[*número da instância*].

Tabela 3: Soluções melhoradas pela metaheurística *iterated local search* para instâncias médias

			Rand_I $n = 100$				
100.4	271282	100.6	270337	100.7	273825	100.8	273546
100.10	273082	100.11	270990	100.13	271817	100.14	269449
100.16	273268	100.17	273041	100.18	271008	100.19	270693
100.20	274805	100.22	272201	100.24	271966		
			Rand_I $n = 150$				
150.2	606463	150.3	605543	150.4	604134	150.5	603812
150.6	603406	150.7	606578	150.9	608265	150.10	608729
150.11	603100	150.12	605551	150.13	605335	150.14	605543
150.15	609047	150.16	606515	150.17	605529	150.18	603662
150.19	603350	150.20	606298	150.21	606997	150.22	605291
150.23	606100	150.25	606182				
			Rand_I $n = 200$				
200.1	1066684	200.2	1068398	200.3	1067029	200.4	1069595
200.5	1068657	200.6	1066519	200.7	1068614	200.8	1071078
200.9	1070387	200.10	1071074	200.11	1066280	200.12	1070209
200.13	1066074	200.14	1069369	200.15	1071884	200.16	1070096
200.17	1069666	200.18	1068427	200.19	1066390	200.20	1070218
200.21	1075089	200.22	1064294	200.23	1068212	200.24	1070115
200.25	1068815						
			Rand_II $n = 200$				
200.2	1061261	200.10	1064358	200.17	1054523		

Tabela 4: Soluções melhoradas pela metaheurística *iterated local search* para instâncias grandes com $n = [500, 2000]$

			$n0500$				
d001-1	60803	d001-2	54658	d001-3	56573	d001-4	60985
d001-5	60515	d005-1	241929	d005-2	240899	d005-3	244396
d005-4	245139	d005-5	242155	d010-1	443350	d010-2	444629
d010-3	444132	d010-4	445456	d010-5	444369	d050-1	2113768
d050-2	2113611	d050-3	2104950	d050-4	2105669	d050-5	2115329
d100-1	6533225	d100-2	6523731	d100-3	6545317	d100-4	6536337
d100-5	6529215						
			$n1000$				
d001-1	215766	d001-2	212037	d001-3	210291	d001-4	219046
d001-5	217248	d005-1	875366	d005-2	879570	d005-3	884017
d005-4	884772	d005-5	878933	d010-1	1633281	d010-2	1651815
d010-3	1652232	d010-4	1646484	d010-5	1647907	d050-1	8119980
d050-2	8119966	d050-3	8117487	d050-4	8118100	d050-5	8111097
d100-1	25806199	d100-2	25808843	d100-3	25841585	d100-4	25856191
d100-5	25806119						
			$n2000$				
d001-1	790721	d001-2	796426	d001-3	769941	d001-4	796144
d001-5	791511	d005-1	3262747	d005-2	3299835	d005-3	3249031
d005-4	3290421	d005-5	3254298	d010-1	6188364	d010-2	6217529
d010-3	6154844	d010-4	6209572	d010-5	6199186	d050-1	31561267
d050-2	31581397	d050-3	31556113	d050-4	31598415	d050-5	31592474
d100-1	102333341	d100-2	102306079	d100-3	102372668	d100-4	102345355
d100-5	102291909						

Tabela 5: Soluções melhoradas pela metaheurística *iterated local search* para instâncias grandes com $n = [3000, 8000]$

			$n3000$				
d001-1	1686574	d001-2	1696656	d001-3	1662139	d001-4	1692282
d001-5	1700715	d005-1	7085409	d005-2	7106837	d005-3	7032557
d005-4	7096230	d005-5	7091771	d010-1	13505833	d010-2	13544650
d010-3	13486699	d010-4	13546904	d010-5	13564427	d050-1	70039563
d050-2	70164149	d050-3	70124887	d050-4	70174882	d050-5	70169619
d100-1	229261334	d100-2	229284913	d100-3	229413144	d100-4	229283818
d100-5	229335120						
			$n4000$				
d001-1	2898905	d001-2	2901425	d001-3	2867157	d001-4	2906063
d001-5	2899240	d005-1	12277592	d005-2	12307905	d005-3	12243911
d005-4	12298342	d005-5	12267501	d010-1	23565955	d010-2	23588469
d010-3	23545805	d010-4	23599767	d010-5	23598376	d050-1	123637301
d050-2	123726161	d050-3	123687316	d050-4	123839144	d050-5	123659615
d100-1	406665010	d100-2	406620843	d100-3	406761373	d100-4	406721874
d100-5	406632618						
			$n8000$				
d001-1	10680342	d001-2	10655501	d001-3	10617576	d001-4	10641997
d001-5	10654212	d005-1	46716235	d005-2	46707588	d005-3	46609791
d005-4	46637268	d005-5	46639196	d010-1	90725595	d010-2	90857565
d010-3	90698804	d010-4	90776093	d010-5	90860720	d050-1	487154175
d050-2	487174194	d050-3	487161306	d050-4	487212326	d050-5	487010190
d100-1	1618561191	d100-2	1618728626	d100-3	1618706167	d100-4	1618806206
d100-5	1618963805						