Adaptive Large Neighborhood Search aplicado no problema CVRP (Capacitated Vehicle Routing Problem) Tópicos em Otimização: Meta-heurísticas 2024/2 Atividade 2

Matheus Saick De Martin[1]

[1]Departamento de Informática – Universidade Federal do Espírito Santo (UFES)
Vitória – ES – Brazil – 17/02/2025

matheus.martin@edu.ufes.br

Abstract. *The **ALNS** has proven to be effective in problems involving vehicle routing (VRP) and its variants, positioning it as a prominent approach for complex combinatorial problems. Therefore, this work aims to apply the ALNS algorithm to the Capacitated Vehicle Routing Problem (CVRP), which is a variation of the VRP where vehicles must respect a maximum load capacity. Finally, this work seeks to present details of the proposed solution, the results obtained, and the conclusions."*

Resumo. *O ALNS tem se mostrado eficaz em problemas que envolvem roteamento de veículos (VRP) e suas variantes, o que o coloca em posição de destaque para problemas complexos de combinatória. Sendo assim, este trabalho tem como objetivo aplicar o algoritmo ALNS ao problema de roteamento de veículos com capacidade (CVRP), o qual é uma variação do VRP em que os veículos devem respeitar uma capacidade máxima de carga. Por fim, este trabalho visa apresentar detalhes da solução proposta, os resultados obtidos e as conclusões.*

1. Introdução

Problemas de otimização combinatória têm se mostrado desafiadores computacionalmente, mas de extrema importância como por exemplo em áreas de logística e transporte. Nesse contexto, surge o algoritmo *Large Neighborhood Search* (**LNS**) proposto por [Shaw 1998], o qual tem desempenhado um papel fundamental nesse tipo de problema. Essa meta-heurística propõe uma abordagem única baseada em destruição e reparação de soluções de modo a explorar eficientemente vizinhanças extensas [Pisinger and Ropke 2010].

O *Adaptive Large Neighborhood Search* (**ALNS**), uma modificação do algoritmo LNS proposto por [Ropke and Pisinger 2006], introduz a ideia de um mecanismo de adaptação no algoritmo, o que faz com que as heurísticas de destruição e reparo sejam escolhidas dinamicamente de forma a priorizar heurísticas com desempenho positivo na instância a ser resolvida, o que promove um equilíbrio dinâmico entre intensificação e diversificação do algoritmo.

Como mostrado em [Pisinger and Ropke 2007], o ALNS tem tido bons resultados e sido amplamente utilizado no problema de roteamento de veículos (VRP) e suas variantes. Nesse contexto, surge o tema desse trabalho, no qual se propõe a fazer uma aplicação

da heurística sobre o problema CVRP (Capacitated Vehicle Routing Problem), uma das variações do VRP. Nessa variante, uma restrição adicional é imposta, os veículos agora precisam respeitar uma capacidade máxima de forma que a soma da demanda dos clientes atendidos pelo veículo não ultrapasse essa quantidade.

Para a realização dos experimentos, instâncias do problema CVRP foram selecionadas da biblioteca *CVRPLib*. Os resultados obtidos demonstraram um ótimo desempenho do algoritmo, atingindo resultados ótimos em diversas instâncias da biblioteca.

Por fim, neste artigo, será apresentado detalhes sobre a solução utilizada, além disso, serão mostrados os resultados obtidos destacando suas particularidades e, ao final, as conclusões acerca do trabalho desenvolvido.

2. Representação da Solução

Neste trabalho, uma solução é representada como uma lista de rotas, em que cada rota é representada por uma lista de clientes. Os clientes são representados por *id's*, que identificam as características do cliente (demanda, distância para outros clientes, etc.). A ordem dos clientes na rota indica a sequência de visitas realizadas pelo veículo atribuído àquela rota. O depósito não foi representado diretamente nessa solução, porém ele é incluído em todos os cálculos necessários. Um exemplo de solução com 10 clientes e 4 veículos é mostrada a seguir, em que o *id* número 0 representa o depósito:

$$[[1, 2, 3], [4, 5, 6], [7, 8], [9, 10]]$$

$$rota_1 : 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0$$
$$rota_2 : 0 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 0$$
$$rota_3 : 0 \rightarrow 7 \rightarrow 8 \rightarrow 0$$
$$rota_4 : 0 \rightarrow 9 \rightarrow 10 \rightarrow 0$$

Figura 1. Representação de uma Solução.

3. Pseudocódigo da Solução

A versão do *ALNS* é híbrida e utiliza o *Simulated Annealing* para implementar o critério de aceitação de uma solução. Na versão implementada, os métodos de destruição e reparação possuem, além dos pesos, **pontuações** (π) e também o **número de utilizações** (θ), ambos são usados para o cálculo do peso utilizado no sorteio da roleta para seleção do par de métodos para aquela iteração. Além disso, uma nova variável chamada **tamanho do segmento** foi introduzida, essa variável indica a cada quantas iterações o peso dos métodos devem ser atualizados. Em síntese, o algoritmo implementado e parte dos valores de hiperparâmetros foram inspirados no trabalho desenvolvido em [Chen et al. 2018].

Algorithm 1 Adaptive large neighborhood search híbrido com Simulated Annealing

input: uma solução inicial válida x e hiperparâmetros.
$x^b = x$; $\rho^- = (1, \ldots, 1)$; $\rho^+ = (1, \ldots, 1)$;
$\pi^- = (0, \ldots, 0)$; $\pi^+ = (0, \ldots, 0)$; $\theta^- = (0, \ldots, 0)$; $\theta^+ = (0, \ldots, 0)$;
$\sigma = [\sigma_1, \sigma_2, \sigma_3]$
$T = T_0$
$iteracao = 0$
Iniciar $tempo$
while $iteracao < iteracao\_maxima$ *e* $tempo < tempo\_maximo$ **do**
    Selecionar métodos de destruição e reparação $d \in \Omega^-$ e $r \in \Omega^+$ usando ρ^- e ρ^+;
    Gerar nova solução x^t utilizando os métodos d e r selecionados
    if $f(x^t) < f(x^b)$ **then**
        $x^b = x^t$;
    end if
    $\Delta f = f(x^t) - f(x)$
    if $y \in [0, 1) < e^{-\Delta f/T}$ **then**
        $x = x^t$;
    end if
    Atualizar pontuações dos métodos π^- e π^+
    Atualizar número de utilizações dos métodos θ^- e θ^+
    if $iteracao \% tamanho\_segmento = 0$ **then**
        Atualizar pesos dos métodos ρ^- e ρ^+
        Zerar π^- e π^+, θ^- e θ^+
    end if
    Atualizar temperatura T
    Atualizar $tempo$
    $iteracoes = iteracoes + 1$
end while
return x^b

A solução inicial que serve como entrada para o algoritmo é gerada através de tentativa e erro, clientes aleatoriamente são inseridos nos veículos até antes de estourar a capacidade, assim todo veículo é preenchido ao máximo com clientes em ordem aleatória. Caso algum cliente fique de fora, a solução é inválida e todo o processo é refeito até uma solução válida ser gerada.

Sabe-se que em problemas similares vários tipos de função objetivo podem ser adotadas até simultaneamente (minimização de distâncias, minimização de rotas, etc.), porém este trabalho se restringe somente a função objetivo que é a minimização da distância entre os clientes em uma mesma rota (incluindo o depósito).

A probabilidade de um método ser escolhido na iteração é mostrada a seguir. Essa probabilidade se refere ao método de seleção da roleta, onde os elementos são ponderados pelo peso (ρ). Como exemplo, tem-se o cálculo de ϕ_j (probabilidade) para um método j de destruição:

$$\phi_j^- = \frac{\rho_j^-}{\sum_{k=1}^{|\Omega^-|} \rho_k^-} \tag{1}$$

Quanto a atualização dos pesos, estes são atualizados seguindo a fórmula abaixo, é possível perceber que pontuações altas favorecem o peso do método, no entanto a pontuação deve ser amortizada pelo número de utilizações. Na fórmula, r regula o quanto do valor original do peso vai ser mantido. Como exemplo, tem-se o cálculo de ρ_j^- para um método j de destruição:

$$\rho_j^- = \begin{cases} r\frac{\pi_j^-}{\theta_j^-} + (1-r)\rho_j^-, & \theta_j^- \neq 0 \\ \rho_j^-, & \theta_j^- = 0 \end{cases} \tag{2}$$

Além disso, a pontuação é atualizada com o valor da função ψ. O valor dessa função vai depender do desempenho do método na iteração. As condições e os valores que podem ser assumidos são descritos na definição da função ψ na fórmula (4). Os valores de σ_1, σ_2 e σ_3 escolhidos nestre trabalho foram: 1.0, 0.4, 0.25. Como exemplo, tem-se o cálculo de π_j^- para um método j de destruição:

$$\pi_j^- = \pi_j^- + \psi_j \tag{3}$$

$$\psi = \max \begin{cases} \sigma_1 & \text{Se a solução é o melhor global} \\ \sigma_2 & \text{Se a solução é melhor que a atual} \\ \sigma_3 & \text{Se a solução foi aceita} \\ \text{Senão} & 0 \end{cases} \tag{4}$$

A função de atualização da temperatura segue a versão logarítmica, essa função é muito utilizada na literatura e possui um bom desempenho geral, por isso foi escolhida para estre trabalho. O valor da temperatura inicial escolhida foi de 1000.

$$T = \frac{T_0}{\log(1 + iteracao)} \tag{5}$$

4. Descrição da Geração de Vizinhanças

Na implementação, foram utilizados 3 operadores de destruição: *random removal*, *worst removal*, *shaw removal*; e 3 operadores de reparação: *greedy insertion*, *regret-2*, *regret-3*. Os destalhes dos operadores são mostrados a seguir.

4.1. Random Removal

Os q clientes são selecionados aleatoriamente para serem removidos da solução, o que tende a gerar um conjunto ruim de clientes removidos para serem inseridos novamente, porém ajuda a diversificar as soluções.

4.2. Worst Removal

Esse método procura remover q clientes de alto custo para posteriormente serem inseridos e possivelmente gerar uma melhor solução. Um cliente é removido se possui o maior custo de remoção, ou seja, calcule o valor da solução com e sem ele, aquele que possuir a maior diferença (gerando o maior custo) será removido primeiro, repita o processo até q clientes forem removidos. Ou seja, para um cliente k, o custo é:

$$custo(i, s) = f(s) - f_{-k}(s)$$

Onde $f_{-k}(s)$ representa o custo da solução com o cliente k removido.

Algorithm 2 Worst Removal

1: **Input:** $s \in \{soluções\}$, $q \in N$, $p \in R_+$
2: **while** $q > 0$ **do**
3: Seja L uma lista com todos os clientes i ordenados de forma decrescente pelo $custo(i, s)$
4: Selecione um número aleatório y no intervalo $[0, 1)$
5: Selecione o cliente $c = L[\lfloor y^p |L| \rfloor]$
6: Remova c da solução s
7: $q = q - 1$
8: **end while**

4.3. Shaw Removal

Esse tipo de abordagem tenta remover clientes que são considerados semelhantes de alguma maneira. A ideia geral é que se clientes parecidos forem removidos, será fácil de fazer trocas entre eles devido à similaridade e dessa forma talvez gerar soluções melhores. Para clientes i e j, a semelhança (*relatedness*) nesse trabalho foi definida como:

$$R_{ij} = \alpha dist_{ij} + \beta |demand_i - demandj|$$

Onde $dist_{ij}$ representa a distância entre dois clientes i,j e $demand_k$ é a demanda correspondente do cliente k. Além disso, os valores das constantes α e β foram definidas como 0.75 e 0.1. Pode-se perceber que a fórmula tenta relacionar clientes que estão perto um do outro e que possuem demandas parecidas. Observe que valores pequenos de R indicam um alto relacionamento.

Algorithm 3 Shaw Removal

1: **Input:** $s \in \{soluções\}$, $q \in N$, $p \in R_+$
2: selecione um cliente aleatório c de S
3: $D = \{c\}$
4: **while** $|D| < q$ **do**
5: Selecione um cliente aleatório c de D
6: Seja L uma lista contendo todos os clientes de s que não estão em D
7: Ordene L de tal forma que $i < j \Rightarrow R(r, L[i]) < R(r, L[j])$
8: Selecione um número aleatório y do intervalo $[0, 1)$
9: $D = D \cup \{L[\lfloor y^p |L| \rfloor]\}$
10: **end while**
11: Remova os clientes em D de s

A variável p utilizada pelo *Worst Removal* e pelo *Shaw Removal* introduz aleatoriedade e foi definida com o valor 3.0 neste trabalho.

4.4. Greedy Insertion

A estratégia de inserção gulosa nesse trabalho varia da versão tradicional. Na implementação deste trabalho, é feita uma iteração sobre a lista de clientes removidos D e insere o cliente na posição de melhor inserção de todas as posições possíveis (posição de menor custo de inserção, esse custo pode ser definido da mesma forma que o custo de remoção mostrado anteriormente). Esse processo é repetido q vezes até todos os clientes tiverem sido inseridos. Isso é diferente da implementação tradicional, pois nela o cliente inserido primeiro é aquele que possui o melhor (ou seja, o menor) custo de inserção dentre todos os clientes. A decisão de fazer diferente foi para aumentar a aleatoriedade e diminuir o custo computacional alto da implementação tradicional.

4.5. Regret-2 e Regret-3

Os últimos dois operadores de reparação são variações de um mesmo operador chamado *regret-k*. Esse operador introduz uma espécie de *lookahead*, em que, para todo cliente, é analisado não só a melhor posição de inserção de todas as possíveis, mas sim as k melhores. Em cada iteração, o cliente escolhido para inserção é aquele que possui a maior diferença entre sua melhor posição e suas outras $k-1$ melhores, quanto maior esse valor, maior é o "arrependimento"de não inserir o cliente no momento. Podemos definir o valor de *regret* para um cliente i com k melhores posições sendo:

$$regret_i = \sum_{j=1}^{k} F_k(i) - F_1(i)$$

Onde $F_k(i)$ representa o custo de inserção do cliente i na sua k-ésima melhor posição de inserção.

5. Detalhes de Implementação

O trabalho foi implementado utilizando a linguagem *python* de programação, o interpretador *python* utilizado foi a versão 3.10.12. A máquina na qual os experimentos foram executados possui sistema operacional *Windows 10 Pro 22H2*, processador *Intel(R) Core(TM) i3-10105 CPU @ 3.70GHz*, uma placa de vídeo *NVIDIA GeForce RTX 4060 8GB VRAM* e 16GB de RAM.

6. Resultados Computacionais

Para realizar os experimentos, foram selecionados testes da biblioteca de problemas *CVRPLib* (`http://vrp.atd-lab.inf.puc-rio.br/index.php/en/`). Mais especificamente, todos os conjuntos de instâncias A, B (Augerat et al) e F (Fisher). O código da implementação do algoritmo pode ser encontrado no repositório github de link: `https://github.com/saick123/T2-Otimizacao-Metaheuristicas-CVRP-com-ALNS`.

Além disso, nos testes, foi considerado que as rotas tem quantidade de rotas fixa, sendo o número de rotas aquele apresentado no arquivo da instância da biblioteca *CVRPLib*. Neste trabalho, todas as distâncias entre clientes ou cliente-depósito foram arredondadas para inteiro.

Para finalizar, nesta seção, serão apresentados os resultados acerca da seleção de hiperparâmetros, resultados dos testes finais e a apresentação de um gráfico de evolução das soluções sobre uma instância específica dos casos de teste.

6.1. Seleção de Hiperparâmetros

Para a etapa de seleção de hiperparâmetros, foram escolhidos o q_{max} e o r para um *Grid Search*, foram testados valores pequenos, médios e grandes para cada um deles. O primeiro hiperparâmetro é o q_{max}, este representa a porcentagem máxima de destruição que pode ser aplicada na solução, ou seja, a porcentagem máxima de clientes que podem ser removidos. Segundamente, o r é utilizado na atualização dos pesos dos métodos e índica o quanto o novo valor do peso vai ser afetado em relação a pontuação do método. Ambos possuem um alto grau de influência no desempenho do algoritmo e por esse motivo foram escolhidos.

Os experimentos desta fase foram feitos somente nas instâncias de teste: A-n32-k5, A-n46-k7 e A-n80-k10, B-n50-k8 e B-n78-k10. Além disso, o algoritmo foi executado somente 1 vez por 300 segundos para cada combinação de valores dos hiperparâmetros.

Na Tabela 1, são apresentadas 4 colunas: nome da instância, valor de q_{max}, valor de r e o valor da função objetivo para melhor solução obtida (f_{mh}). Além do mais, de acordo com a Tabela 1, pode-se perceber que o conjunto de valores $q_{max} = 0.15$ e $r = 0.3$ obtiveram os melhores resultados mais vezes, portanto foram escolhidos para os testes finais.

6.2. Evolução Temporal de uma Solução

Nesta seção, é apresentada um gráfico na Figura 2, no qual podemos observar a evolução de uma solução durante o programa. O gráfico foi gerado sobre uma execução da instância *A-n32-k5* por 300 segundos usando os valores dos hiperparâmetros escolhidos na seção anterior. Em verde, há uma linha horizontal tracejada sobre o valor do melhor custo encontrado, além disso, temos um ponto verde no gráfico que mostra a iteração (114160) e o melhor valor de custo (784) encontrado pela primeira vez.

Deste modo, é possível observar que o algoritmo implementado converge em soluções boas rapidamente, isso se deve aos seus operadores que são em grande parte gulosos. De fato, na instância em questão, o algoritmo encontrou o melhor valor possível para a instância em poucas iterações.

6.3. Resultados dos Testes

Os testes realizados nessa seção foram executados, para cada instância, por 300 segundos com repetição de 5 vezes. A Tabela 2, na qual os resultados são mostrados, possui 6 colunas: nome da instância, melhor valor da função objetivo conhecido para a instância ($f_{Sol-Otima}$), o melhor e médio valor da função objetivo encontrado nas repetições (f_{mh-min} e $f_{mh-medio}$), o menor e médio tempo para encontrar a melhor solução nas repetições ($Tempo_{min}$ e $Tempo_{medio}$) e o gap_{min} e gap_{medio} em relação ao melhor valor da função objetivo conhecido.

Os resultados obtidos mostram que a implementação da metaheurística ALNS apresentada desempenha muito bem nas instâncias testadas. Esse fato pode ser atestado pelos valores da função objetivo da melhor solução encontrada que são muito próximos do

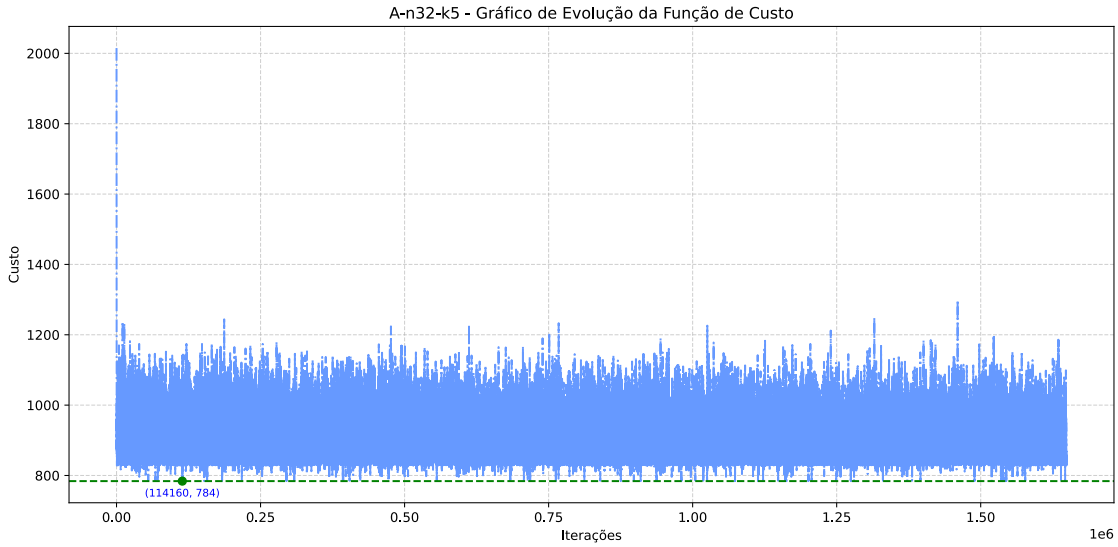

Figura 2. Descrição da imagem

melhor valor ótimo conhecido na maioria das instâncias. Ademais, é possível observar na tabela que, em 28 das instâncias testadas e destacadas em negrito, o valor ótimo conhecido foi alcançado, o que mostra mais uma vez a robustez do algoritmo em achar boas soluções.

Ao longo do trabalho, um dos desafios encontrados foi em relação a aproximação das distâncias entre nós (clientes e depósito). No site da biblioteca, nenhuma informação sobre o esse assunto é dada, porém é evidente ao testar as instâncias que algum tipo de aproximação foi feito. No atual trabalho, foi feito o arredondamento das distâncias para inteiro, no entanto o caso *F-n45-k4* acabou tendo um custo menor que o ótimo. Assim, é pertinente que os autores da biblioteca especificassem melhor sua metodologia em seu site, o que garantiria uma padronização metodológica para as instâncias lá disponibilizadas.

7. Conclusões

Neste trabalho, foi explorada uma implementação da metaheurística *ALNS* ao problema *CVRP*. A implementação gerou bons resultados chegando até a alcançar os melhores resultados conhecidos no conjunto em até 28 instâncias. Desta forma, além dos trabalhos citados anteriormente, esse trabalho mostra mais uma vez o bom desempenho do ALNS no problema *VRP* e suas variantes. Seus operadores de destruição e reparação de características gulosas baseadas em boas heurísticas permitem o algoritmo encontrar boas soluções rapidamente, garantindo uma convergência rápida em boas soluções.

Para trabalhos futuros, busca-se explorar implementações que permitem soluções inválidas alterando a função objetivo para realizar uma penalização, o que pode provocar uma melhor busca no espaço de soluções. Além disso, estudos de funções de reparação que garantem a construção de uma solução válida para o problema podem ser explorados com o intuito de diminuir o custo computacional relacionado a reparação de soluções.

Referências

Chen, S., Chen, R., Wang, G.-G., Gao, J., and Sangaiah, A. K. (2018). An adaptive large neighborhood search heuristic for dynamic vehicle routing problems. *Computers & Electrical Engineering*, 67:596–607.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435.

Pisinger, D. and Ropke, S. (2010). *Large Neighborhood Search*, pages 399–419.

Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40:455–472.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In Maher, M. and Puget, J.-F., editors, *Principles and Practice of Constraint Programming — CP98*, pages 417–431, Berlin, Heidelberg. Springer Berlin Heidelberg.

Instância	q_{max}	r	f_{mh}
A-n32-k5	0.15	0.1	784
A-n32-k5	0.15	0.3	784
A-n32-k5	0.15	0.5	784
A-n32-k5	0.2	0.1	784
A-n32-k5	0.2	0.3	784
A-n32-k5	0.2	0.5	784
A-n32-k5	0.3	0.1	784
A-n32-k5	0.3	0.3	784
A-n32-k5	0.3	0.5	784
A-n46-k7	0.15	0.1	917
A-n46-k7	**0.15**	**0.3**	914
A-n46-k7	0.15	0.5	920
A-n46-k7	0.2	0.1	915
A-n46-k7	0.2	0.3	917
A-n46-k7	0.2	0.5	915
A-n46-k7	0.3	0.1	917
A-n46-k7	0.3	0.3	921
A-n46-k7	0.3	0.5	920
A-n80-k10	0.15	0.1	1851
A-n80-k10	0.15	0.3	1847
A-n80-k10	0.15	0.5	1852
A-n80-k10	**0.2**	**0.1**	1839
A-n80-k10	0.2	0.3	1859
A-n80-k10	0.2	0.5	1846
A-n80-k10	0.3	0.1	1845
A-n80-k10	0.3	0.3	1874
A-n80-k10	0.3	0.5	1849
B-n50-k8	0.15	0.1	1327
B-n50-k8	**0.15**	**0.3**	1321
B-n50-k8	**0.15**	**0.5**	1321
B-n50-k8	0.2	0.1	1326
B-n50-k8	0.2	0.3	1325
B-n50-k8	0.2	0.5	1325
B-n50-k8	0.3	0.1	1329
B-n50-k8	0.3	0.3	1326
B-n50-k8	0.3	0.5	1327
B-n78-k10	0.15	0.1	1283
B-n78-k10	**0.15**	**0.3**	1277
B-n78-k10	0.15	0.5	1287
B-n78-k10	0.2	0.1	1291
B-n78-k10	0.2	0.3	1286
B-n78-k10	0.2	0.5	1284
B-n78-k10	0.3	0.1	1283
B-n78-k10	0.3	0.3	1289
B-n78-k10	0.3	0.5	1293

Tabela 1. Resultados dos testes de *Seleção de Hiperparâmetros* nas instâncias A-n32-k5, A-n46-k7 e A-n80-k10, B-n50-k8 e B-n78-k10.

Instância	$f_{Sol-Otima}$	f_{mh}		Tempo(s)		gap_{min}	gap_{medio}
		Min	*Média*	*Min*	*Média*		
A-n32-k5	784	784	784.00	0.116	6.146	0.0000000	0.0000000
A-n33-k5	661	661	661.00	0.082	1.095	0.0000000	0.0000000
A-n33-k6	742	742	742.00	1.091	26.246	0.0000000	0.0000000
A-n34-k5	778	778	778.00	1.616	11.133	0.0000000	0.0000000
A-n36-k5	799	799	799.00	95.114	167.541	0.0000000	0.0000000
A-n37-k5	669	669	669.00	0.171	0.679	0.0000000	0.0000000
A-n37-k6	949	949	949.20	30.852	128.554	0.0000000	0.0002107
A-n38-k5	730	730	730.00	1.208	8.681	0.0000000	0.0000000
A-n39-k5	822	822	822.00	30.733	124.760	0.0000000	0.0000000
A-n39-k6	831	831	831.40	2.350	80.621	0.0000000	0.0004813
A-n44-k6	937	939	940.20	44.729	129.460	0.0021345	0.0034152
A-n45-k6	944	944	946.80	40.422	121.827	0.0000000	0.0029661
A-n45-k7	1146	1146	1151.00	28.530	154.252	0.0000000	0.0043630
A-n46-k7	914	914	915.80	15.101	129.334	0.0000000	0.0019694
A-n48-k7	1073	1073	1077.40	55.623	114.615	0.0000000	0.0041007
A-n53-k7	1010	1010	1012.40	33.210	112.217	0.0000000	0.0023762
A-n54-k7	1167	1174	1178.60	62.124	166.684	0.0059983	0.0099400
A-n55-k9	1073	1073	1075.00	11.099	155.972	0.0000000	0.0018639
A-n60-k9	1354	1374	1382.00	49.712	151.834	0.0147710	0.0206795
A-n61-k9	1034	1046	1057.80	6.144	187.034	0.0116054	0.0230174
A-n62-k8	1288	1312	1315.00	63.546	178.649	0.0186335	0.0209627
A-n63-k10	1314	1341	1345.60	35.914	150.332	0.0205479	0.0240487
A-n63-k9	1616	1660	1666.00	48.726	81.061	0.0272277	0.0309406
A-n64-k9	1401	1452	1456.00	48.037	133.996	0.0364026	0.0392577
A-n65-k9	1174	1177	1181.00	70.495	170.450	0.0025554	0.0059625
A-n69-k9	1159	1171	1173.20	138.744	205.307	0.0103538	0.0122519
A-n80-k10	1763	1820	1846.40	38.792	146.176	0.0323313	0.0473057
B-n31-k5	672	672	673.40	5.714	45.654	0.0000000	0.0020833
B-n34-k5	788	788	788.00	1.534	2.282	0.0000000	0.0000000
B-n35-k5	955	955	955.00	2.375	25.938	0.0000000	0.0000000
B-n38-k6	805	805	805.80	0.787	39.534	0.0000000	0.0009938
B-n39-k5	549	549	549.40	8.638	133.823	0.0000000	0.0007286
B-n41-k6	829	829	829.40	85.732	129.471	0.0000000	0.0004825
B-n43-k6	742	742	742.00	4.174	130.150	0.0000000	0.0000000
B-n44-k7	909	909	911.60	88.245	140.793	0.0000000	0.0028603
B-n45-k5	751	753	757.20	53.992	155.138	0.0026631	0.0082557
B-n45-k6	678	680	686.80	14.759	92.254	0.0029499	0.0129794
B-n50-k7	741	741	741.00	1.231	3.262	0.0000000	0.0000000
B-n50-k8	1312	1324	1326.60	38.734	171.951	0.0091463	0.0111280
B-n51-k7	1032	1039	1041.20	107.875	214.851	0.0067829	0.0089147
B-n52-k7	747	747	747.00	28.685	69.430	0.0000000	0.0000000
B-n56-k7	707	713	713.80	27.073	39.177	0.0084866	0.0096181
B-n57-k7	1153	1186	1211.40	124.840	204.916	0.0286210	0.0506505
B-n57-k9	1598	1615	1619.00	20.307	78.198	0.0106383	0.0131414
B-n63-k10	1496	1538	1547.00	116.334	214.443	0.0280749	0.0340909
B-n64-k9	861	871	875.20	56.490	154.717	0.0116144	0.0164925
B-n66-k9	1316	1339	1343.20	12.091	105.483	0.0174772	0.0206687
B-n67-k10	1032	1055	1057.80	24.193	178.493	0.0222868	0.0250000
B-n68-k9	1272	1306	1310.40	3.606	140.542	0.0267296	0.0301887
B-n78-k10	1221	1256	1278.80	166.927	204.578	0.0286650	0.0473382
F-n135-k7	1162	1176	1184.80	14.890	119.765	0.0120482	0.0196213
F-n45-k4	724	721	721.00	1.241	8.262	-0.0041436	-0.0041436
F-n72-k4	237	237	238.40	25.372	142.366	0.0000000	0.0059072

Tabela 2. Resultados dos testes realizados os conjuntos de instâncias A, B (Augerat et al) e F (Fisher).