UNIVERSIDADE ESTADUAL DE MARINGÁ
DEPARTAMENTO DE INFORMÁTICA
Curso: Ciência da Computação
Disciplina: 6903 - Modelagem e Otimização Algorítmica
Professor: Ademir Aparecido Constantino

Avaliação 2

Implementação de Algoritmos Heurísticos

Equipe:

Gabriel Henrique Costanzi RA: 102573

Thiago Yuji Yoshimura Yamamoto RA: 112649

Maringá - PR
2022

1) Introdução

O objetivo deste trabalho é praticar a implementação de algoritmos heurísticos de construção e melhoria, aplicadas no problema do caixeiro viajante. Para isso, foi implementado e testado quatro algoritmos: dois de construção e dois de melhoria.

Os algoritmos heurísticos são utilizados quando os métodos clássicos demoram muito tempo para encontrar uma solução. Eles não garantem encontrar a solução ótima de um problema, mas entregam uma solução boa em uma quantidade de tempo razoável.

Os métodos construtivos são utilizados para obter uma solução inicial para o problema por meio da adição de novos elementos em uma solução vazia a cada iteração. Após a obtenção de uma solução inicial por meio do método construtivo, torna-se necessário melhorar a qualidade do mesmo, para isso utiliza-se os métodos de busca heurísticos de melhoria, no qual o algoritmo navega pelas soluções vizinhas à procura de uma melhoria na solução.

2) Descrição do Problema

O Problema do Caixeiro Viajante (PCV) tem como problema definir um circuito com o menor custo de n cidades, que se inicia e encerra sua viagem na primeira cidade, sendo que deve-se visitá-los apenas uma única vez, não importando a ordem e que de cada uma delas pode-se ir diretamente a qualquer outra.

3) Descrição do Algoritmo

3.1) Heurístico Construtivo Vizinho mais Próximo

O algoritmo heurístico construtivo do vizinho mais próximo tem como complexidade $O(n^2)$, pois para gerar a solução inicial constrói-se uma rota adicionando, a cada passo, o vértice mais próximo do último vértice inserido, sendo que para encontra-lo temos que verificar todos os vértices. Para que não seja necessário verificar se o vértice já foi visitado, retiramos ele da lista de vértices e colocamos na lista de saída (solução).

3.2) Heurístico Construtivo Inserção rápida

O algoritmo heurístico construtivo da inserção rápida tem complexidade $O(n^2)$ e é utilizado para gerar uma solução inicial para o problema do caixeiro viajante. O algoritmo primeiramente forma um ciclo inicial com 1 vértice aleatório do grafo. Após isso escolhe-se um vértice aleatório já presente no ciclo e encontra-se o vértice que ainda não faz parte do ciclo e que é o mais próximo do vértice escolhido. Após isso, insere-se o vértice selecionado no ciclo na posição posterior ao vértice escolhido. A etapa anterior é repetida até que não existam mais vértices para serem adicionados no ciclo.

3.3) Heurístico Melhorativo 2-OPT

O algoritmo heurístico melhorativo 2-OPT implementado no trabalho tem complexidade de

$mn + mn^2$, ou seja, $O(mn^2)$, onde m é o número de melhorias e n o número de vértices.

Para cada uma das n arestas da solução inicial criam-se soluções vizinhas com todas as n-2 arestas que não são adjacentes a ela trocando-se os vértices entre as arestas e invertendo a lista dos vértices entre elas. A primeira solução vizinha que produz uma melhora no resultado é adotada e o ciclo se repete até que não se possa mais fazer melhorias na solução. A procura dos vizinhos da solução inicial é feita de forma sequencial a partir dos laços de repetição "for" consecutivos, e representamos as arestas que estão sendo verificadas em uma determinada iteração de AB e CD.

Como o algoritmo procura o vizinho de peso menor do que o original de forma sequencial, a probabilidade de melhoria também se acumulam de forma sequencial, ou seja, ao encontrar uma melhoria na aresta AB da posição A, o algoritmo 2-opt será executado novamente, reposicionando o vértice pivô para o início sendo necessário verificar $(A-1)^2$ vértices da lista, caso a melhoria esteja depois do vértice A.

Para contornar esse problema, foi implementado o método que apelidamos de "Fator de Aceleração", no qual consiste no reposicionamento da aresta pivô do início para a posição da aresta que houve melhoria (pulando os $(A-1)^2$ verificações), e caso nenhuma melhoria seja encontrado no intervalo de A até o fim da lista verifica-se no intervalo de 0 até A-1. Desta forma não estamos ignorando os outros casos ($(A-1)^2$ possibilidades), apenas ajudamos o algoritmo encontrar as melhorias mais rapidamente. Essa melhoria funciona apenas para algoritmos que utilizam o método first improvement, pois o encontro da solução otimizada encerra a execução do 2-opt.

3.4) Heurístico Melhorativo 3-OPT

O algoritmo heurístico melhorativo 3-OPT implementado no trabalho tem como complexidade de $mn + mn^3$, ou seja, $O(mn^3)$, onde m é o número de melhorias e n o número de vértices.

Uma execução do algoritmo 3-OPT remove 3 arestas do caminho e reconecta-os gerando 7 caminhos diferentes, cada uma delas são analisadas para verificar se o novo caminho possui o valor otimizado, este procedimento é executado para todas as combinações de arestas (n^3 combinações). Se houver otimização, o novo caminho é construído a partir da reconstrução do caminho original com os vértices trocados, fazendo necessário inverter uma sub lista entre os dois vértices trocados (a inversão de uma lista tem custo O(n)), seguindo a mesma lógica do algoritmo 2-OPT.

Como apenas uma execução do 3-OPT não garante a melhor solução local, deve-se executar o algoritmo até que todas as soluções vizinhas não resultam em uma otimização (executa-se o 3-OPT toda vez que há uma melhoria, ou seja, m vezes). Considerando as informações anteriores a complexidade do algoritmo seria de $O(m*n^3)$.

Da mesma forma que o 2-opt, o 3-opt também possui o problema da procura dos vizinhos

de forma sequencial, e como utilizamos o método first improvement aplicamos o método “Fator de Aceleração” nas três arestas, porém mesmo utilizando esse método o número de iterações continua grande. Para contornar este problema, a cada ciclo a aresta AB recebe a posição da última melhoria, e diferentemente do 2-opt que a aresta AB (pivô) retorna para o ínicio caso não encontra-se uma melhora, o 3-opt manterá a sua posição original até que alcance a posição final da lista. Isso ocorre de forma semelhante na aresta EF, a diferença está no reposicionamento da aresta EF na posição equivalente a 1/10 do intervalo entre CD e EF.

3) Resultados

Os resultados dos testes a seguir foram gerados pelos algoritmos 2-OPT e 3-OPT com o método “Fator de Aceleração”, sendo assim, o algoritmo de 2-OPT garante ser a melhor solução local, porém o 3-OPT não consegue obter a mesma garantia (pelo fato de não verificar todos os vizinhos). A justificativa do seu uso é a longa duração do tempo de processamento, com o uso do método o tempo máximo de processamento foi de 2 horas no teste pla85900.
Como meio de comparação do tempo de processamento dos algoritmos com e sem melhoramento, a tabela abaixo demonstrará o tempo de execução dos algoritmos sem o método “Fator de Aceleração”, baseado no teste com 1002 e 4461 vértices, (pr1002.tsp e fnl4461.tsp) do 2-OPT e 3-OPT de forma **aproximada**, com a solução inicial provida do construtivo vizinho mais próximo.

Tabela 1 - Tempo de execução sem melhoria

Nome do teste	**Tempo de execução (em segundos)**	
	2-OPT	**3-OPT**
pr1002	1,17	158,40
fnl4461	108,30	52.417,94
brd14051	2492,67	4.465.802,25
pla33810	20.466,81	89.698.464,46
pla85900	381.481,90	3.332.329.388,03

Vale ressaltar que o algoritmo foi implementado na linguagem Java, a IDE Visual Studio Code e executado em uma máquina com as especificações:

- Processador: Ryzen 5 5600x
- 16GB RAM

3.1) PCV - pr1002.tsp

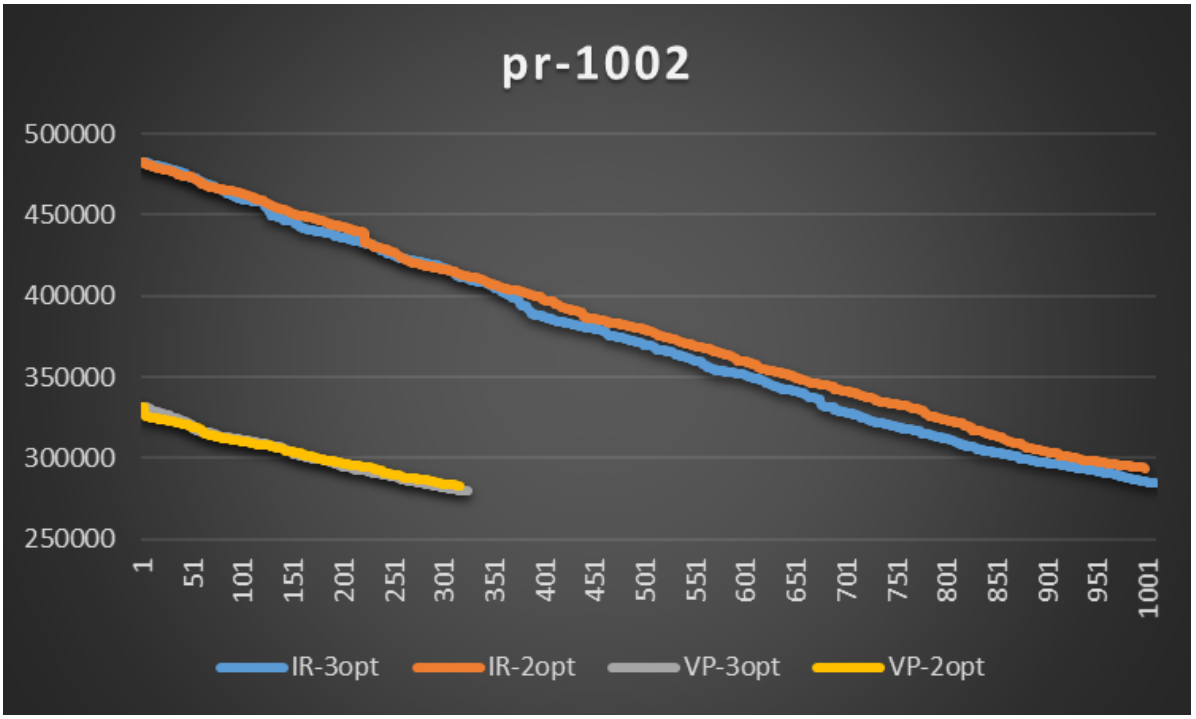
Tabela 2 - Construtivo (pr1002)

Construtivo - pr1002	
Algoritmo	**Peso Total**
Vizinho mais Próximo (VP)	331831
Inserção Rápida (IR)	483432

Tabela 3 - Melhorativo (pr1002)

Melhorativo - pr1002		
Algoritmo	**Peso Total**	**Precisão**
IR + 2-OPT	293916	88,13%
IR + 3-OPT	284696	90,99%
VP + 2-OPT	283036	91,52%
VP + 3-OPT	279436	92,70%

Figura 1 - Gráfico da melhoria dos pesos do pr1002



3.2) PCV - fnl4461.tsp

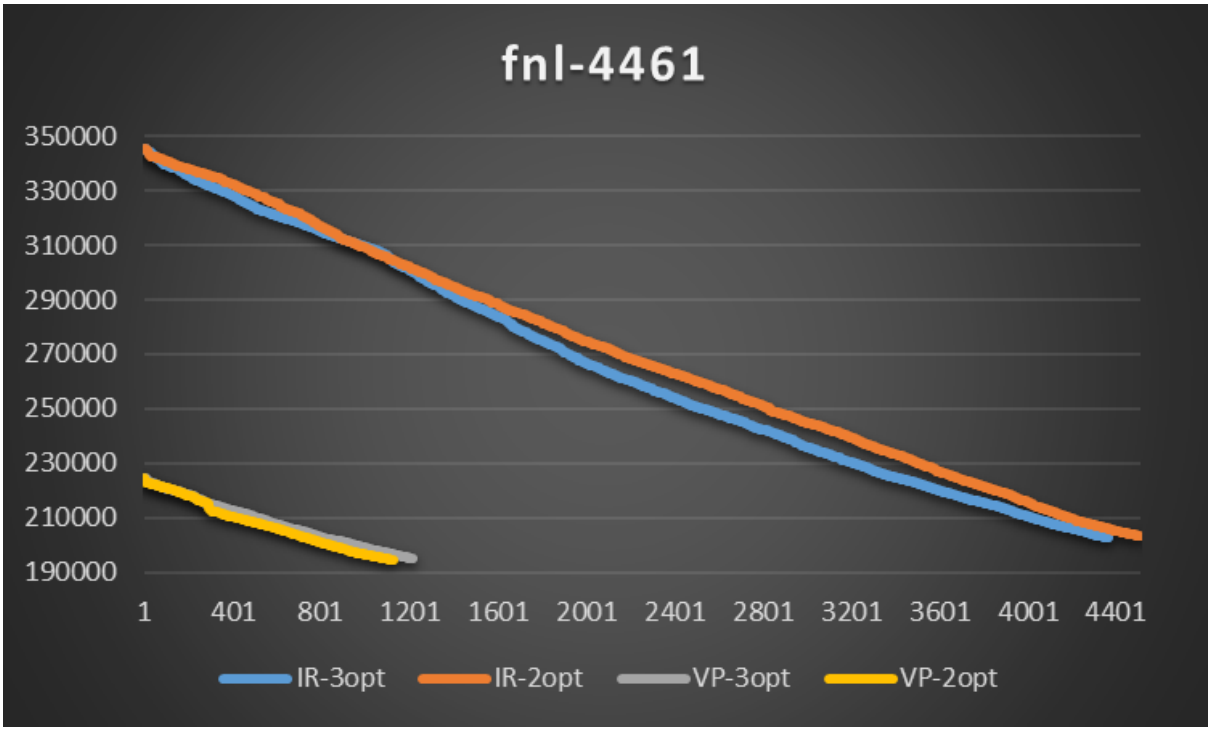
Tabela 4 - Construtivo (fnl4461)

Construtivo - fnl4461	
Algoritmo	**Peso Total**
Vizinho mais Próximo (VP)	224671
Inserção Rápida (IR)	345498

Tabela 5 - Melhorativo (fnl4461)

Melhorativo - fnl4461		
Algoritmo	**Peso Total**	**Precisão**
IR + 2-OPT	203311	89,79%
IR + 3-OPT	202766	90,03%
VP + 2-OPT	194413	93,90%
VP + 3-OPT	194965	93,64%

Figura 2 - Gráfico da melhoria dos pesos do fnl4461



3.3) PCV - pla7397.tsp

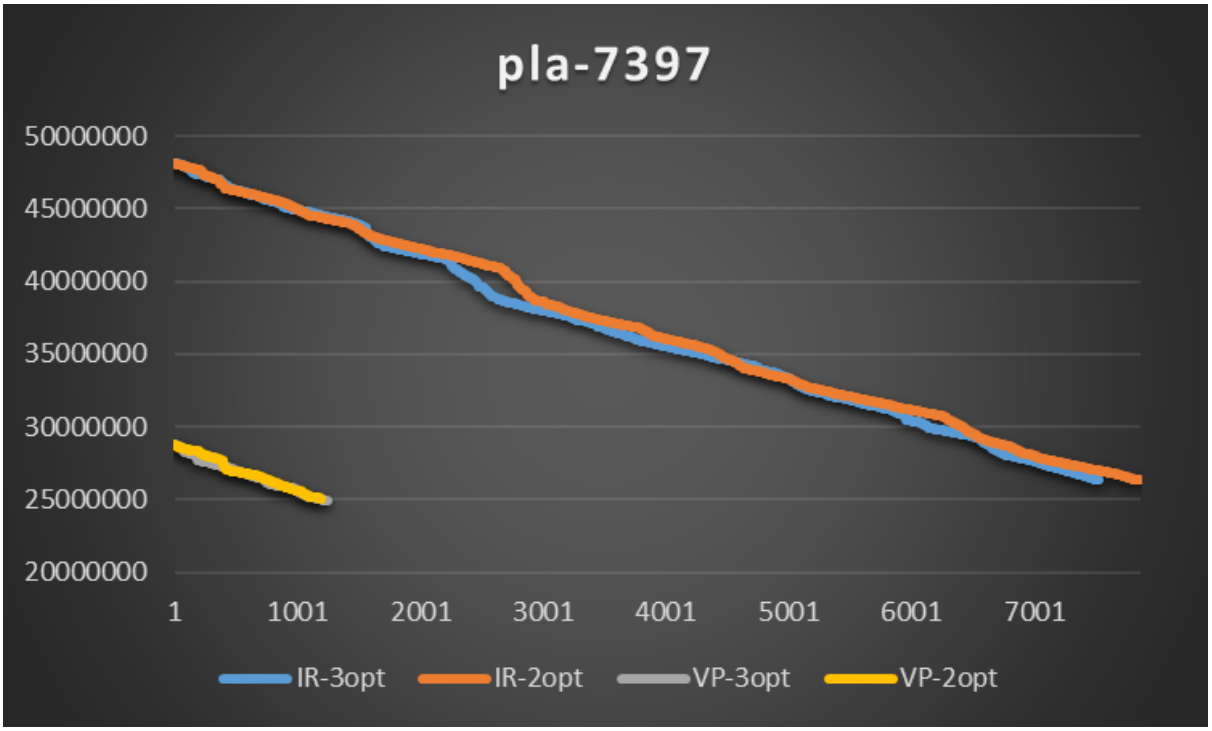
Tabela 6 - Construtivo (pla7397)

Construtivo - pla7397	
Algoritmo	**Peso Total**
Vizinho mais Próximo (VP)	28780453
Inserção Rápida (IR)	48169169

Tabela 7 - Melhorativo (pla7397)

Melhorativo - pla7397		
Algoritmo	**Peso Total**	**Precisão**
IR + 2-OPT	26328949	88,34%
IR + 3-OPT	26384349	88,16%
VP + 2-OPT	25098869	92,67%
VP + 3-OPT	24937101	93,27%

Figura 3 - Gráfico da melhoria dos pesos do pla7397



3.4) PCV - brd14051.tsp

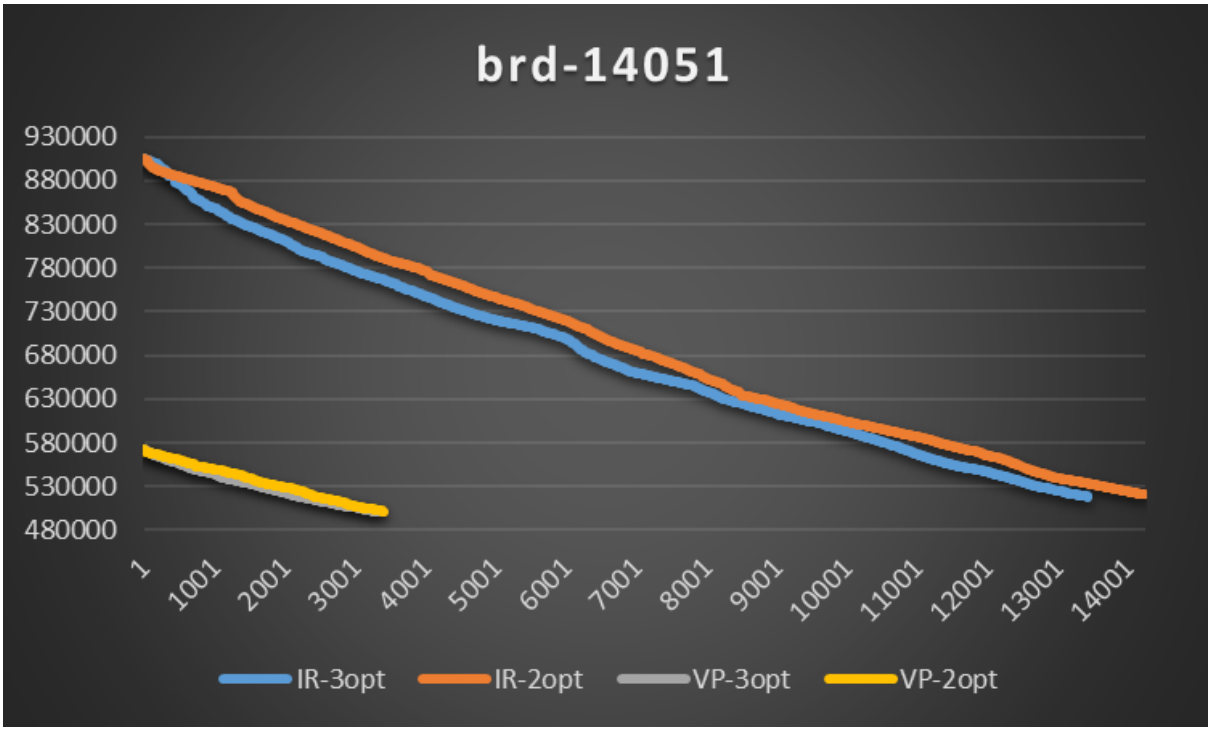
Tabela 8 - Construtivo (brd14051)

Construtivo - brd14051	
Algoritmo	**Peso Total**
Vizinho mais Próximo (VP)	572012
Inserção Rápida (IR)	904671

Tabela 9 - Melhorativo (brd14051)

Melhorativo - brd14051		
Algoritmo	**Peso Total**	**Precisão**
IR + 2-OPT	520362	90,20%
IR + 3-OPT	518357	90,55%
VP + 2-OPT	501661	93,56%
VP + 3-OPT	500542	93,77%

Figura 4 - Gráfico da melhoria dos pesos do brd14051



3.5) PCV - d15112.tsp

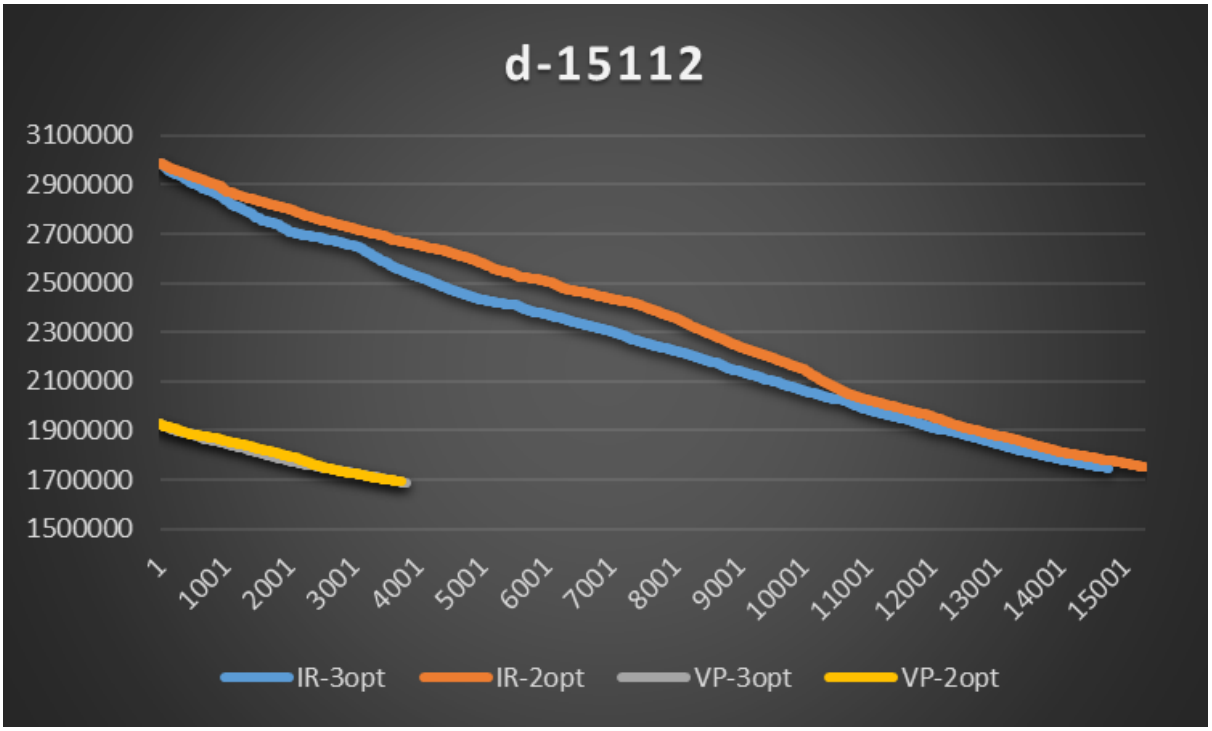
Tabela 10 - Construtivo (d15112)

Construtivo - d15112	
Algoritmo	**Peso Total**
Vizinho mais Próximo (VP)	1926597
Inserção Rápida (IR)	2987979

Tabela 11 - Melhorativo (d15112)

Melhorativo - d15112		
Algoritmo	**Peso Total**	**Precisão**
IR + 2-OPT	1751254	89,82%
IR + 3-OPT	1748955	89,94%
VP + 2-OPT	1691189	93,01%
VP + 3-OPT	1687790	93,20%

Figura 5 - Gráfico da melhoria dos pesos do d15112



3.6) PCV - d18512.tsp

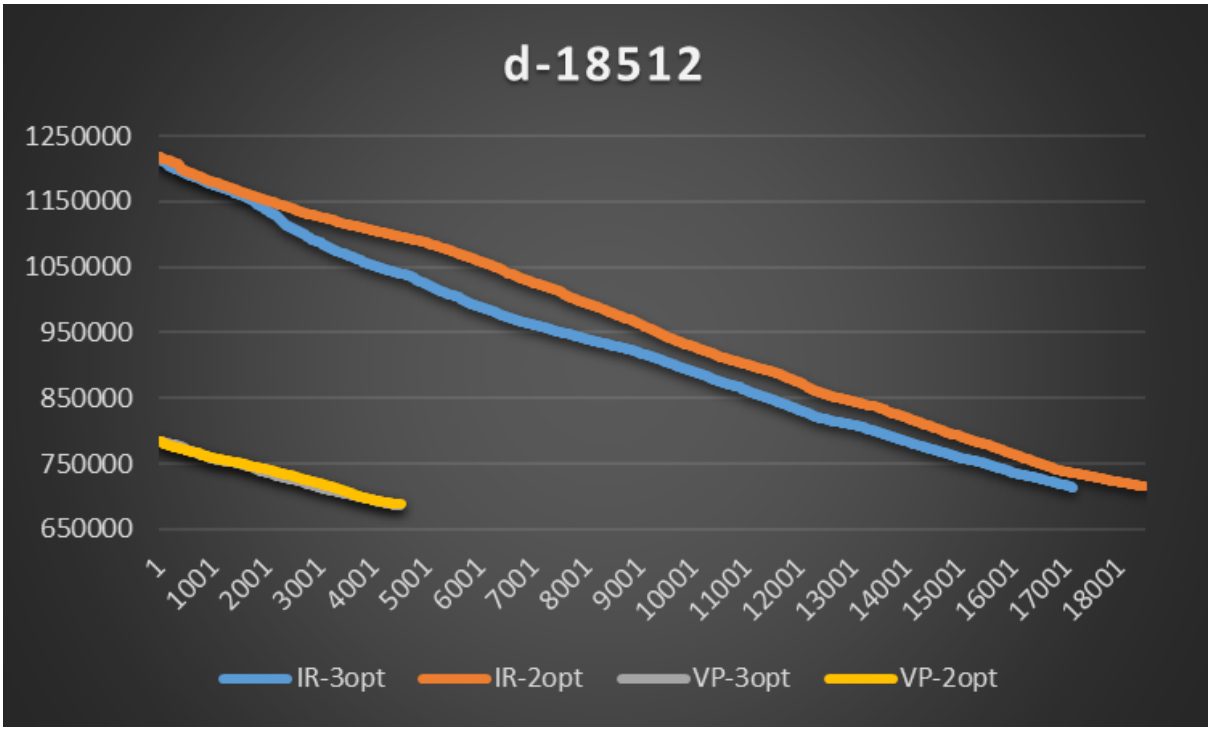
Tabela 12 - Construtivo (d18512)

Construtivo - d18512	
Algoritmo	**Peso Total**
Vizinho mais Próximo (VP)	785057
Inserção Rápida (IR)	1217953

Tabela 13 - Melhorativo (d18512)

Melhorativo - d18512		
Algoritmo	**Peso Total**	**Precisão**
IR + 2-OPT	715027	90,24%
IR + 3-OPT	713967	90,37%
VP + 2-OPT	687867	93,80%
VP + 3-OPT	685777	94,08%

Figura 6 - Gráfico da melhoria dos pesos do d18512



3.7) PCV - pla33810.tsp

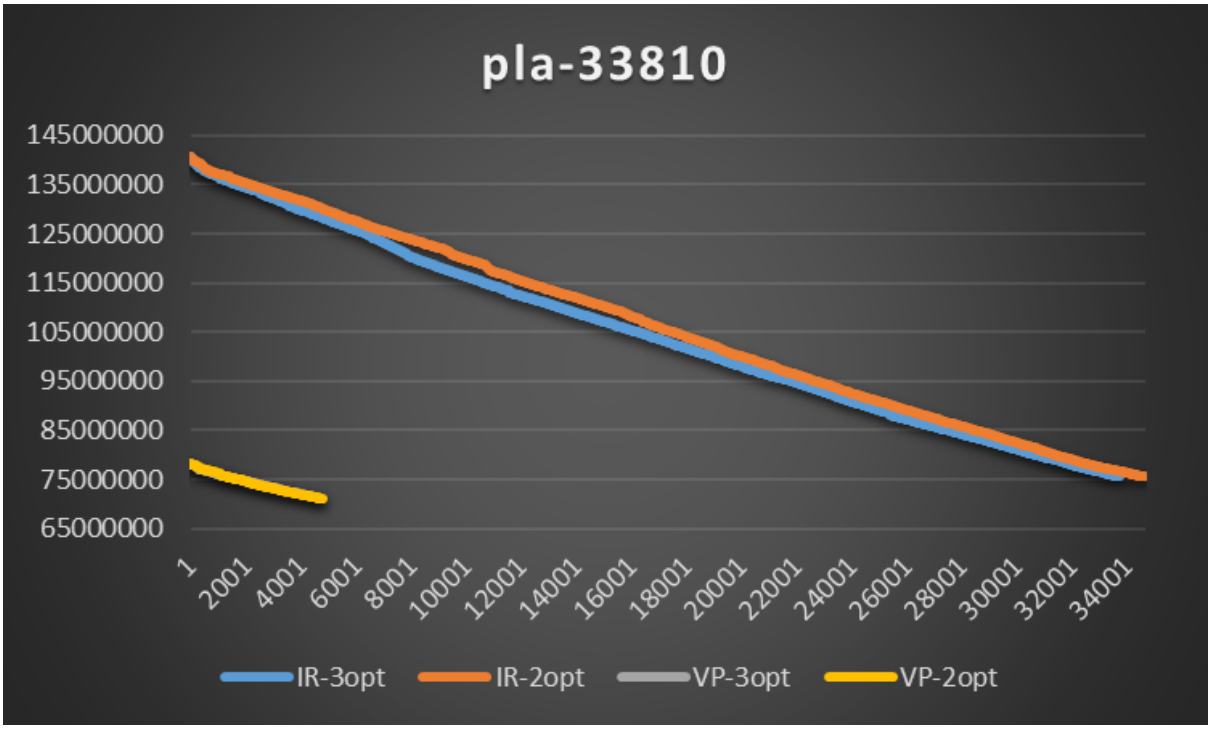
Tabela 14 - Construtivo (pla33810)

Construtivo - pla33810	
Algoritmo	**Peso Total**
Vizinho mais Próximo (VP)	78178141
Inserção Rápida (IR)	140727459

Tabela 15 - Melhorativo (pla33810)

Melhorativo - pla33810		
Algoritmo	**Peso Total**	**Precisão**
IR + 2-OPT	75530693	87,44%
IR + 3-OPT	75568252	87,40%
VP + 2-OPT	71067031	92,93%
VP + 3-OPT	71123446	92,86%

Figura 7 - Gráfico da melhoria dos pesos do 33810



3.8) PCV - pla85900.tsp

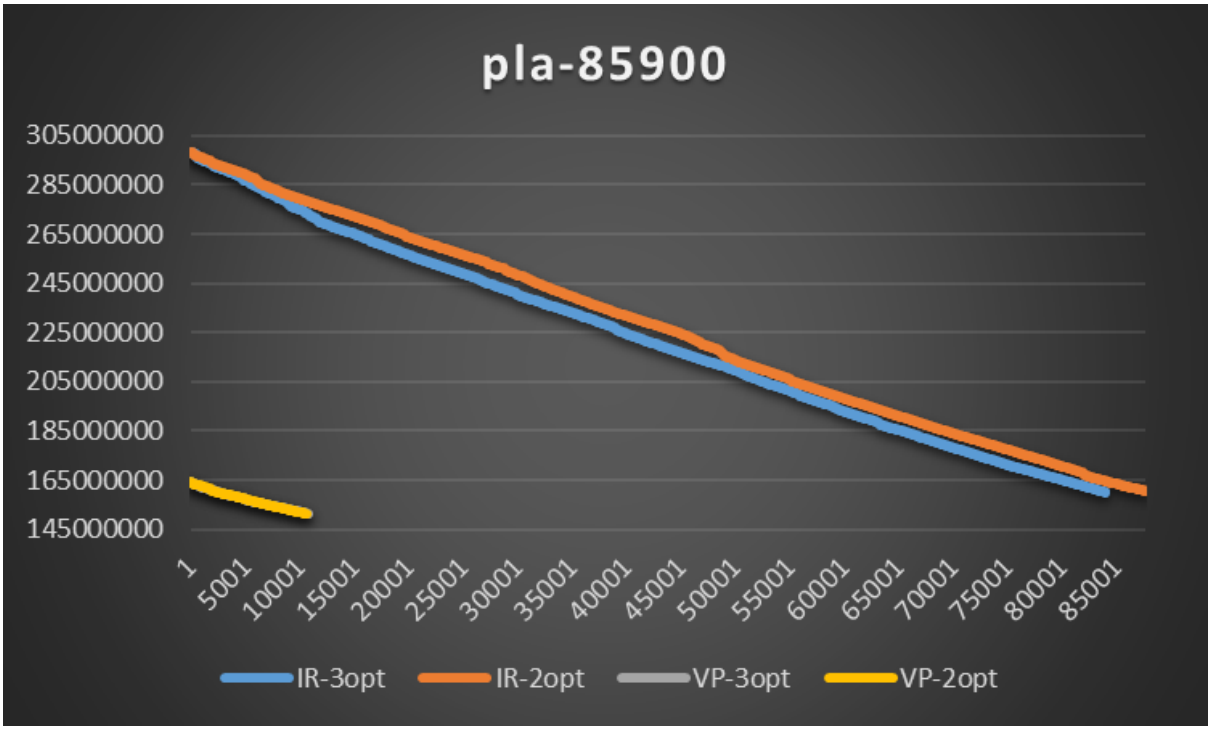
Tabela 16 - Construtivo (pla85900)

Construtivo - pla85900	
Algoritmo	**Peso Total**
Vizinho mais Próximo (VP)	164168160
Inserção Rápida (IR)	298432100

Tabela 17 - Melhorativo (pla85900)

Melhorativo - pla85900		
Algoritmo	**Peso Total**	**Precisão**
IR + 2-OPT	160499202	89,71%
IR + 3-OPT	160102434	89,98%
VP + 2-OPT	151378176	94,05%
VP + 3-OPT	151349267	94,07%

Figura 8 - Gráfico da melhoria dos pesos do pla85900



4) Conclusão dos Resultados

Como é possível observar nas tabelas construtivas e nos gráficos, o algoritmo construtivo do vizinho mais próximo gera soluções melhores do que o algoritmo de inserção rápida, principalmente devido à escolha de um vértice aleatório do ciclo durante a etapa de inserção dos vértices restantes.

A partir da análise dos resultados obtidos, a melhora da solução inicial provida do construtivo de inserção rápida é pior comparada ao do vizinho mais próximo, isto se dá pelo fato de que, quanto mais melhorias são feitas, mais rápido encontra-se a solução ótima local.

Como podemos observar nos gráficos das figuras, o algoritmo 3-OPT a cada ciclo encontra uma melhoria maior comparado ao algoritmo 2-OPT, porém ao analisar o resultado final o peso total do 3-OPT em alguns casos é pior, isso ocorre pelo fato da aplicação das restrições das arestas AB e EF resultarem na não verificação de todas as possibilidades de troca.