Proposta de solução do Problema de Caixeiro Viajante através dos métodos *Greedy Randomized Adaptive Search Procedure* e *Interated Local Search*

Elizabeth Akemi Fujito, Pedro de Mendonça Maia, Roberto Proença Silva

Resumo – Este trabalho propõe um modelo de metaheurística para solucionar o Problema do Caixeiro Viajante (PCV), sendo que o objetivo principal é de minimizar o tempo (em horas) para o caixeiro percorrer todas as cidades, passando uma única vez em cada uma delas. Para solucionar este problema, testamos dois métodos: *Interated Local Search* (ILS) e o *Greedy Randomized Adaptive Search Procedure*(GRASP). Utilizamos a heurística construtiva *Greedy Procedure* para obter a solução inicial, e realizamos a busca local com o algoritmo 3-opt *improvement*.

***Index Terms*—Busca local, GRASP, ILS, metaheurística, PCV.**

I. Introdução

O Problema do Caixeiro Viajante (PCV) é provavelmente o problema mais famoso e extensivamente estudado no campo da otimização combinatória. É um problema bem conhecido na área de rede e pesquisa operacional. A simplicidade e complexidade do PCV têm atraído a atenção de muitos pesquisadores e sua importância se deve ao fato de serem tão difíceis de serem resolvidos, mas muito utilizados para modelar diversos problemas do mundo real [1].

Para definir o PCV, considere um conjunto de nós V, e um conjunto A de arcos todos conectados aos nós V. Seja $t_{i,j}$ o tempo de percurso do arco $a_{i,j} \in A$, ou seja: o tempo gasto para ir do nó $v_i \in V$ ao nó $v_j \in V$. O problema do PCV será encontrar o menor tempo para percorrer o circuito Hamiltoniano no grafo $G(V, A)$. Neste trabalho, o caixeiro terá que percorrer 250 cidades no menor tempo possível, saindo da cidade 1 e retornando à mesma após visitar uma única vez as outras 249 cidades, concluindo assim a rota.

Apesar do PCV ser conceitualmente simples, não é fácil obter uma solução ideal para o problema. Havendo n cidades e considerando um grafo completo, qualquer permutação de n nós produzirão uma solução possível, ou seja, existem $n!$ rotas possíveis na região de busca. Isso impede que métodos exatos sejam utilizados em problemas grandes e por isso muitos pesquisadores buscam métodos alternativos que resolvam instâncias grandes do problema em um tempo razoável e que encontrem soluções satisfatórias [1].

Métodos clássicos para solucionar o PCV com algoritmos exatos resultam em complexidades computacionais exponenciais. Assim, novos métodos surgiram para preencher esta lacuna, utilizando diferentes tipos de técnicas de otimização: algoritmos de otimização baseados na natureza, algoritmos de otimização baseados em populações e outros. Esses métodos denominados metaheurísticas apresentam melhor desempenho do que os métodos exato e o heurístico, sendo muito utilizados por pesquisadores e cientistas na resolução de problemas de otimização combinatória. Devido ao uso do conceito de randomização na busca por melhores soluções, a metaheurística é mais efetiva no escape do ótimo local, podendo obter soluções de melhor qualidade. O emprego de hibridização pode melhorar a qualidade da resolução de problemas de optimização, se comparados com a heurística ou metaheurística. [1].

II. Problema do Caixeiro Viajante

O PCV é um dos problemas aplicados no mundo real que consiste em um número de cidades que devem ser visitadas a partir da cidade inicial, por apenas um vendedor; sendo que cada cidade é visitada uma única vez. Fig. 1.

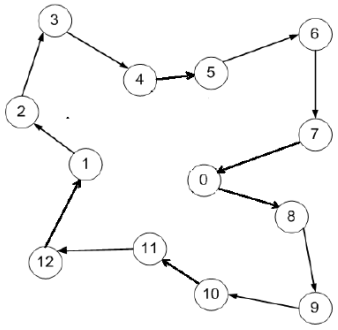

Figura 1. Um exemplo de rota do PCV

Seja $G(V, A)$ um grafo orientado completo, onde $V = 1, ..., n$ consiste de um conjunto de n cidades e A representa um conjunto de arcos $A = (i, j) : i, j \in V i \neq j$. O problema foi definido para grafos assimétricos, isto é, quando os tempo $t_{i,j}$ e $t_{j,i}$ de percorrer as arestas $a_{i,j}$ e $a_{j,i}$ são diferentes. A solução do PCV é determinar um caminho que o caixeiro fará este percurso com o menor tempo total. As variáveis $x_{i,j}$ são usadas para representar os caminhos que o caixeiro usará, sendo que:

$$x_{i,j} = \begin{cases} 1, & \text{se o caminho da cidade } i \text{ para a } j \text{ é usado} \\ 0, & \text{caso contrário} \end{cases}$$

O modelo completo de otimização combinatória dado por [2] é apresentado a seguir:

$$min \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} t_{i,j} x_{i,j} \quad (1)$$

$$\sum_{i=1}^{n} x_{i,j} = 1 \quad j = 1, ..., n \quad j \neq i \quad (2)$$

$$\sum_{j=1}^{n} x_{i,j} = 1 \quad i = 1, ..., n \quad i \neq j \quad (3)$$

$$\sum_{i \in S} \sum_{j \in S, i \neq j} x_{i,j} \leq |S| - 1 \quad S \subset V \quad 2 \leq |S| \leq \left[\frac{n}{2}\right] \quad (4)$$

$$x_{i,j} \in [0, 1] \quad \forall i \in [1, ..., n] \quad \forall j \in [1, ..., n] \quad i \neq j \quad (5)$$

- A função objetivo (1) minimiza o tempo total percorrido em uma viagem.
- O conjunto de restrições (2) garante que o caixeiro chegue apenas uma vez em cada nó.
- O conjunto de restrições (3) garante que o caixeiro saia apenas uma vez em cada nó.
- O conjunto de restrições (4) evita a presença de sub-rotas.
- O conjunto de restrições (5) define o limite das variáveis de decisão.

III. Modelagem

Como mencionado anteriormente, o Problema do Caixeiro Viajante é um problema de solução complicada e por se tratar de uma instância grande, se tornou necessário abrir mão da otimalidade para obter resultados em tempo viável. Nesse sentido, uma abordagem diferente é utilizada para as soluções candidatas e são propostas meta-heurísticas para resolver o problema.

A. Modelo da solução candidata

Uma solução candidata é chamada de rota. A rota é representada no modelo como um vetor de N inteiros distintos. A sequencia desse vetor representa a ordem de cidades visitadas pelo caixeiro viajante. A Fig. 2 mostra um exemplo com 7 cidades.

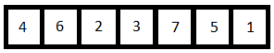

Figura 2. Exemplo de rota seguida pelo caixeiro viajante

Nesse exemplo, ele inicia a rota pela cidade 4, depois passará pela 6, em seguida para a 2, e assim por diante até chegar à cidade 1. Após passar pela última cidade (no exemplo a 1) ele retorna à cidade inicial (no exemplo a 4), fechando um ciclo.

O tempo total dessa rota é calculado a partir do somatório entre os nós adjacentes do vetor. O tempo entre o nó final e o inicial também é somado para fechar o ciclo.

Por se tratar de um ciclo, é importante ressaltar que a cidade por onde se inicia a rota não é relevante, apenas a ordem importa. A rota figura 2 é idêntica à rota 3-7-5-1-4-6-2, por exemplo.

Como neste trabalho o caixeiro deve sempre partir da cidade 1, a solução final é dada deslocando a cidade 1 até o início do vetor. No caso do exemplo, essa rota seria 1-4-6-2-3-7-5.

IV. Algoritmos utilizados

Meta-heurísticas costumam apresentar desempenho satisfatório na solução de problemas combinatoriais, portanto decidiu-se testar dois métodos para a resolução do PCV: o GRASP e o ILS.

A. GRASP

O GRASP (*Greedy Randomized Adaptive Search Procedure*) é uma metaheurística constituída por heurísticas construtivas e busca local[3]. Ele consiste de múltiplas aplicações de busca local, cada uma iniciando de uma solução diferente[7]. A geração da solução inicial é baseada nas três primeiras iniciais de sua sigla em inglês: gulosa (*Greedy*), aleatória (*Randomized*) e adaptativa (*Adaptative*).

Este algoritmo é composto de duas fases: construção da solução inicial e aplicação da busca local na solução construída, na tentativa de melhorar a qualidade da solução[5].

O método GRASP amostra o espaço com gerações rapidamente. Quanto melhor for a qualidade da solução gerada na primeira fase, maior será a velocidade para encontrar um ótimo local pela fase de busca local. O método GRASP é simples, rápido e pode ser facilmente integrado com outras técnicas de busca[7].

A seguir é apresentado o pseudocódigo do GRASP[6].

Algorithm 1 Algorithm GRASP: Max_Iterations, Seed

```
Read_Input();
for k=1,...,Max_Iterations do
  Solution ← Greedy_Randomized_Construction(Seed)
  if Solution is not feasible then
    Solution ← Repair(Solution);
  end if
  Solution ← Local_Search(Solution);
  Update_Solution(Solution, Best_Solution);
end for
return Best_Solution;
end GRASP
```

O método de criação da solução inicial e os algoritmos de busca local usados no trabalho são apresentados a seguir.

1) Construção da Solução Inicial (Greedy Randomized Construction)

A estratégia de construção de uma solução no GRASP consiste na definição de um critério de avaliação dos elementos que podem ser inseridos em um conjunto, e que, ao final do processo será uma solução para o problema de otimização que se pretende resolver[3]. Nesta primeira fase, uma solução é construida elemento a elemento até que ele represente uma solução viável para o problema. Inicialmente esses elementos estão em uma lista de candidatos (LC). Através do parâmetro α, cria-se a lista RCL(*Restricted Candidate List*) contendo os melhores elementos de LC[7].

O tamanho de RCL é determinado por:

$$CardinalidadeRCL = \alpha * CardinalidadeLC$$

A lista RLC é composta com os melhores elementos, conforme segue:

$$c(e) \in [c^{mim}, c^{mim} + \alpha(c^{max} - c^{mim})]$$

Onde:

- c(e) é o custo incremental associado à inclusão do elemento e à solução parcial;
- c^{min} e c^{max} são respectivamente o menor e o maior custos incrementais.

O valor de α ifluencia na qualidade e diversidade da solução gerada na fase de construção.

α = 0 : construção puramente gulosa;
α = 1 : construção puramente aleatória.

Valores baixos para α geram soluções gulosas de boa qualidade, mas com pouca diversidade. Um valor alto para α, próximo a cardialidade de LC leva a uma grande diversidade com soluções de qualidade inferior. Isso também influencia o processo de busca local, pois soluções de qualidade inferior tornam o processo de busca local mais lento. O valor de α pode ser constante, mas pode também sofrer alterações a cada iteração ou por meio de um esquema aleatório ou adaptativo[7].

Após a definição da RLC, seleciona-se um elemento da mesma para compor a solução. A seleção do elemento pode ser realizada aleatoriamente ou através de um critério guloso. Esses dois tipos de seleções provocam duas variações do GRASP conforme construção da solução: aleatório ou guloso. Após a adição do elemento na solução, o processo continua com a atualização de ambas as listas LC e RLC. O processo de construção é finalizado quando a cardinalidade de LC possuir valor zero[7].

2) Algoritmo de Busca Local

O algoritmo de busca local básico é inicializado a partir de uma solução viável. Ele define, para cada solução, uma vizinhança composta por um conjunto de soluções com características "muito próximas". Dada uma solução corrente, uma das formas de implementar um algoritmo de busca local é percorrer a vizinhança dessa solução em busca de outra com um melhor valor. Caso esta solução vizinha seja encontrada, torna-se a nova solução corrente e o algoritmo continua. Do contrário, a solução corrente é um ótimo local em relação à vizinhança adotada [8].

O algoritmo 2 apresenta um pseudocódigo genérico para métodos de busca local[6].

2.1) ***2-opt***

Este é o método discreto de busca local mais simples para o PCV, onde a vizinhança $N(S)$ da solução S definida pelo conjunto de soluções pode ser alcançada através da permutação de duas arestas não-adjacentes de S. Este movimento é denominado 2-*interchange* e é demonstrado na Fig.3.

Algorithm 2 Algorithm Local_Search(Solution)

```
while Solution is not locally optimal do
  Find s' ∈ N (solution) with f(s') > f(solution);
  Solution ← s';
end while
return Solution
end Local_Search
```

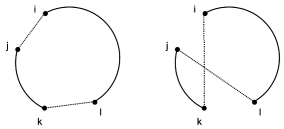

Figura 3. Exemplo de uma operação 2-opt

O algoritmo testa a cada iteração algumas combinações de permutação de aresta e caso alguma solução encontrada seja melhor que a corrente encontrada, ela é adotada e o algoritmo passa para a iteração seguinte. O fim da busca é alcançada quando nenhuma das permutações realizadas em determinada iteração supera a solução corrente.

Existem algumas variações do algoritmo

- *first improvement*: a cada iteração todas as permutações possíveis vão sendo testadas até que uma solução melhor é encontrada e é adotada imediatamente.
- *best improvement*: a cada iteração todas as permutações são testadas e a melhor solução é adotada ao final dos testes.

2.2) ***3-opt***

Este método é semelhante ao 2-opt, a diferença é que a vizinhança $N(S)$ da solução S definida pelo conjunto de soluções pode ser alcançada através da permutação de três arestas não-adjacentes em S. Este movimento é demonstrado na Fig.4.

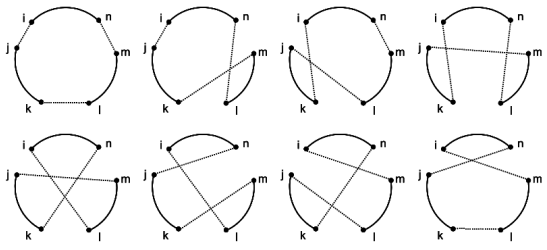

Figura 4. Exemplo de uma operação 3-opt

Nesse caso, além da solução original existem outras 7 formas de religar as arestas removidas. A cada iteração do método todas as combinações são testadas para cada grupo de arestas removidas.

B. ILS

O ILS (*Iterated Local Search*) é um algoritmo heurístico que se baseia no princípio de que um procedimento de busca local

pode ser melhorado, a partir da geração de novas soluções iniciais, que são obtidas através de perturbações em uma solução ótima local. Estas perturbações permitirão a exploração de diferentes soluções pela busca local e evitará um reinício aleatório. Desta forma, o ILS realiza a busca em um pequeno subespaço definido por soluções que são ótimos locais, e não a busca em um espaço completo de soluções[8]. A Fig. 5 mostra o princípio do ILS[6].

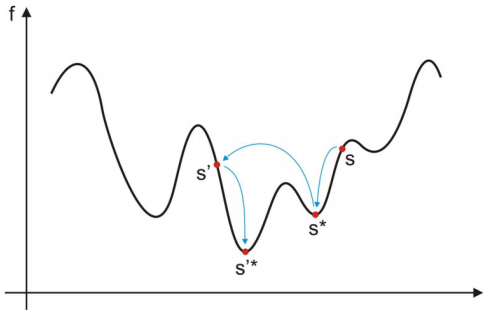

Figura 5. Princípio do ILS

O algoritmo 3 mostra o procedimento geral do ILS[6]:

Algorithm 3 Algorithm Iterated Local Search

s_0 = GenerateInitialSolution
s^* = LocalSearch (s_0)
repeat
  s'= Perturbation (s^*, history)
  s^*'= LocalSearch (s')
  s^* = AcceptanceCriterion (s^*,s^*', history)
until termination condition met

O algoritmo retorna a melhor solução encontrada durante toda a execução.

O ILS depende fortemente de uma boa solução inicial para gerar uma boa solução ótima local, portanto neste trabalho utilizamos o mesmo procedimento adotado no GRASP, o *Greedy Randomized Construction*. Os mecanismos de busca local adotados no ILS também foram os mesmos do GRASP, o 2-opt e o 3-opt. Entretanto o 3-opt apresentou melhor solução.

Os métodos utilizados nos demais passos do ILS são explicados a seguir

1) Perturbação

A perturbação modifica a solução corrente, por meio de movimentos aleatórios de remoção de nós na solução corrente. Esses movimentos são responsáveis por alterar a solução corrente, guiando-a para uma solução intermediária. Dessa forma, o mecanismo de perturbação deve ser forte o suficiente para permitir escapar do ótimo local corrente e permitir também a exploração de diferentes regiões do espaço de soluções. Ao mesmo tempo, a perturbação precisa ser fraca o suficiente para guardar características do ótimo local corrente, evitando o reinício aleatório [3]. No método ILS utilizamos o algoritmo *cross-exchange* para realizar as perturbações.

O *cross-exchange* é um caso específico com 4-opt. Nele, retira-se 4 arcos e religa-se os vértices formando uma cruz, criando desta forma uma nova rota, como mostra a Fig.6. Um aspecto importante desse operador que o torna útil para a perturbação é que o religamento não pode ser desfeito pela busca local 2-opt, 3-opt ou Lin Kernighan[5].

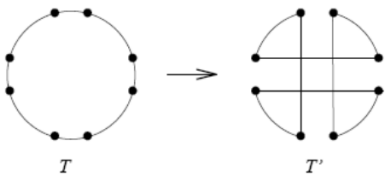

Figura 6. Figura extraida de Applegate et all, Finding tours inthe TSP

2) Critério de aceitação

O Critério de Aceitação define se a solução ótima local retornada pela busca local será aceita ou descartada. Uma solução é aceita se ela melhorar a melhor solução atual. Na heurística ILS, soluções que pioram a melhor solução atual podem também ser aceitas com uma pequena probabilidade. Dessa forma, evita-se a utilização constante da melhor solução atual e, consequentemente, uma rápida estagnação das soluções avaliadas[8].

No trabalho o critério de aceitação adotado foi que após *n* iterações sem melhoria na solução, então o algoritmo aceita uma solução pior.

V. Testes Computacionais

Os testes foram realizados inicialmente com os dois algoritmos. Para cada um, foram realizadas cinco execuções de 30 minutos ou 200 iterações.

Ambos utilizaram o *Greedy Randomized Construction* para inicializar as soluções com o valor de $\alpha = 0.001$, pois haviam muitas distâncias com valores muito pequenos e iguais, então o valor de referência de $\alpha = 0.2$ gerou soluções muito aleatórias. O método de Busca Local adotado também foi o mesmo para os dois casos. Foi usado o 3-opt em 2500 amostras aleatórias em cada busca.

O ILS usou o algoritmo cross-exchange para fazer as perturbações e adotamos como critério de aceitação 20 iterações sem melhorias na solução.

Ao final de cada execução dos métodos ILS e GRASP, o algoritmo 2-opt *best improvement* foi executado em cada um deles, a fim de garantir que obtivemos o mínimo local, uma vez que os métodos utilizados na Busca Local não testavam todas as combinações.

Após terminar esses testes iniciais, analisamos qual obteve o melhor resultado e repetimos as cinco execuções somente para o algoritmo vencedor, porém com o limite de duas horas de execução ou 1000 iterações.

O trabalho não considera todas as rotas válidas. Existem algumas arestas nos dados de entrada com tempo zero. Nesses casos, ao invés de considerar o tempo nulo, é considerada uma aresta inválida. Qualquer solução criada com aresta inexistente em qualquer momento do algoritmo é descartada.

VI. Resultados

A. Resultados iniciais do ILS e GRASP

Como descrito anteriormente, a primeira etapa dos testes consistiu em executar os dois algoritmos cinco vezes e ver qual apresentaria o melhor resultado.

As tabelas I, II e III mostram os resultados das execuções dos algoritmos.

Tabela I
Tempo das melhores rotas encontradas pelos algoritmos

Execução	GRASP	ILS
1	29.9	31.1
2	29.7	30.9
3	30.1	30.4
4	30.4	31.0
5	30.0	32.6

O GRASP apresentou resultados melhores em todas as execuções, como mostra a tabela I. Na execução com pior resultado, o GRASP encontrou uma rota de 30.4h, enquanto o melhor resultado encontrado pelo ILS foi uma rota de 31h.

Tabela II
Tempo de execução dos algoritmos (s)

Execução	GRASP	ILS
1	1830.5	1822.4
2	1812.0	1836.1
3	1810.1	1838.4
4	1814.6	1820.8
5	1809.1	1824.8

A tabela II mostra que os dois algoritmos terminaram as execuções em todos os casos no critério de parada de tempo, pois todas as execuções excederam 1800s.

Tabela III
Número de iteração dos algoritmos

Execução	GRASP	ILS
1	152	138
2	151	137
3	159	130
4	163	157
5	155	146

A tabela III mostra que o GRASP executou em média mais iterações que o ILS em 30 minutos, média de 156 iterações do GRASP contra 141.6 do ILS. Isso mostra que, apesar de recomeçar sempre de um ponto aleatório, o GRASP foi capaz de encontrar mínimos locais mais rápido que o ILS, o que pode justificar o seu desempenho superior.

Um fator que pode ter prejudicado o desempenho do ILS é o método de perturbação, que o fazia gerar soluções muito aleatórias ou muito próximas. A geração de soluções muito aleatórias reinicializa a busca do zero, o que atrasa o processo de busca local, enquanto a geração de soluções próximas não tira o algoritmo de determinada bacia de atração, levando ao mesmo pronto de mínimo local encontrado anteriormente.

B. Resultados Finais do GRASP

Como pôde ser observado nos testes anteriores, o GRASP obteve resultados melhores que o ILS e por isso ele foi escolhido para buscar as soluções finais do PCV.

Nesse teste o GRASP foi executado mais cinco vezes com as mesmas heurísticas construtiva e de busca local, porém cada execução teve um limite de 2h ou 1000 iterações. Ao final de cada execução o 2-opt *best improvement* também foi executado para garantir que o mínimo local da melhor solução fosse encontrado.

A tabela IV mostra os resultados das cinco execuções.

Tabela IV
Resultado das execuções finais do GRASP

Execução	Solução	Iterações	Tempo execução (s)
1	30.4	636	7216.1
2	29.4	611	7229.7
3	30.2	637	7202.5
4	29.9	623	7205.7
5	29.4	607	7206.9

Ao analisar a tabela IV é possível perceber que novamente o final das execuções ocorreu quando o critério de parada de tempo de execução foi alcançado. Nesses testes a média de iterações foi de 622.8, o que levou a resultados ligeiramente melhores.

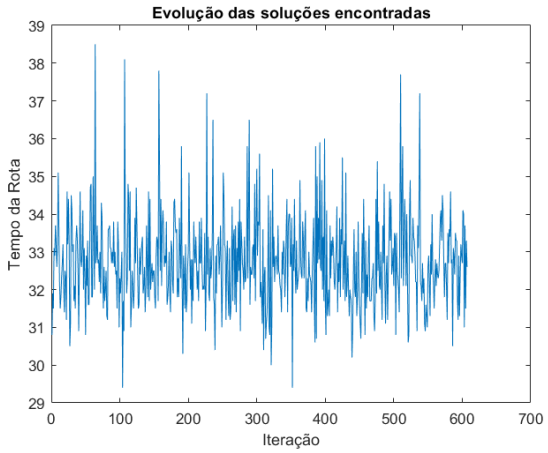

Figura 7. Histórico de soluções da melhor solução do GRASP

Por se tratar de um método simples e aleatório, não é possível saber quando a melhor solução é encontrada. Na Fig. 7, por exemplo, o algoritmo executou mais de 600 iterações, tendo encontrado a melhor solução já na iteração 104.

Comparando as soluções obtidas pelo GRASP entre as execuções de 30 minutos (tabela I) e de duas horas (tabela IV) pode-se perceber que a melhor solução da execução de duas horas apresentou apenas um pequeno ganho em relação a solução de trinta minutos (29.4 contra 29.7). Isso mostra como é importante avaliar a relação custo-benefício entre tempo e qualidade em problemas difíceis, principalmente quando heurísticas são empregadas.

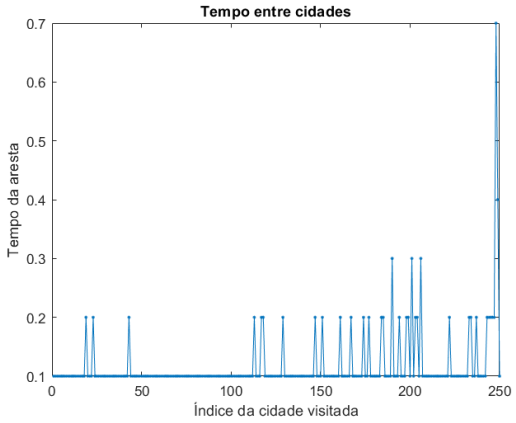

Figura 8. Tempo das arestas usadas da melhor solução do GRASP

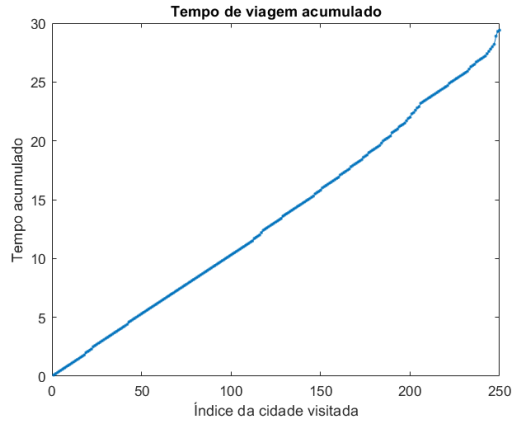

Figura 9. Tempo acumulado da melhor rota encontrada pelo GRASP

A Fig.8 mostra o tempo de viagem entre cada cidade considerando a ordem ideal encontrada. Sabemos que o menor tempo possível é de 0.1h, e observando o gráfico é possível perceber que a maioria das arestas usadas possui o tempo 0.1h. Apenas duas arestas possuem tempo de viagem maior que 0.3h, o que evidencia que a solução encontrada está próxima da melhor solução possível. Esse resultado leva ao tempo acumulado aproximadamente linear como visto na Fig.9.

C. Testes com o grafo completo

Foi realizada uma nova rodada de testes, agora considerando um grafo completo. Isso significa que nesse teste todas as arestas do problema, mesmo a com tempo de viagem igual a zero, são válidas. Ao fazer isso, espera-se que haja uma redução no tempo total das rotas encontradas.

Todos os testes foram realizados novamente, e os resultados da comparação entre o ILS e o GRASP após trinta minutos de execução estão mostrados nas tabelas V, VI e VII.

Como esperado, as soluções encontradas nos nos dois algoritmos foram bem melhores que as anteriores, apresentadas na tabela I. Os resultados do GRASP foram novamente superiores, coincidentemente a pior solução do GRASP foi novamente igual a melhor solução do ILS.

Tabela V
TEMPO DAS MELHORES ROTAS ENCONTRADAS PELOS ALGORITMOS CONSIDERANDO ARESTAS COM TEMPO ZERO

Execução	GRASP	ILS
1	21.8	21.9
2	21.6	24.9
3	21.3	23.8
4	21.9	23.7
5	22.2	24.1

Tabela VI
TEMPO DE EXECUÇÃO DOS ALGORITMOS CONSIDERANDO ARESTAS COM TEMPO ZERO (S)

Execução	GRASP	ILS
1	1801.6	1802.9
2	1801.6	1806.9
3	1800.6	1804.4
4	1801.9	1806.4
5	1802.3	1808.6

Tabela VII
NÚMERO DE ITERAÇÃO DOS ALGORITMOS CONSIDERANDO ARESTAS COM TEMPO ZERO

Execução	GRASP	ILS
1	1244	769
2	1260	720
3	1225	691
4	1222	729
5	1251	750

Ao comparar a tabela VII com a tabela III, percebemos que os dois algoritmos executaram muito mais interações do que no teste inicial. Isso acontece pois no novo teste não foi necessário verificar se uma nova solução é válida. O número de iterações do GRASP foi novamente superior ao do ILS, o que evidencia que o provável problema da perturbação se manteve.

Visto que o GRASP obteve as melhores soluções, foram realizadas mais cinco execuções de duas horas desse algoritmo. Os resultados podem ser vistos na tabela VIII.

Tabela VIII
RESULTADO DAS EXECUÇÕES FINAIS DO GRASP CONSIDERANDO ARESTAS COM TEMPO ZERO

Execução	Solução	Iterações	Tempo execução (s)
1	21.3	4976	7201.8
2	21.3	5011	7204.4
3	21.4	5007	7202.3
4	21.3	5052	7204.7
5	21.4	4608	7207.1

Os resultados mostraram que o tempo de viagem da melhor solução caiu muito em relação ao teste inicial (tabela IV), como esperado. O número de iterações também foi muito superior ao inicial.

Outro resultado que se repetiu foi o pequeno ganho na melhor solução do GRASP executado por duas horas em relação à execução por trinta minutos. Esse resultado corrobora para a discussão já levantada sobre o custo-benefício entre qualidade e tempo ao se trabalhar com heurísticas. A melhor solução encontrada em trinta minutos coincide com a melhor

solução encontrada em duas horas, porém ao executar por duas horas esse resultado ou outro muito próximo foi alcançado em todos os casos.

VII. Conclusão

Após a realização do trabalho, percebemos que realmente o PCV é um problema muito fácil de ser entendido, mas com solução extremamente desafiadora. Foram empregadas duas meta-heurísticas para a solução do problema e seus desempenhos foram comparados.

Após a realização de testes, verificamos que o método GRASP apresentou uma melhor solução ótima, bem como um melhor tempo de processamento, conforme verificado nos experimentos computacionais. Esses resultados mostraram como a configuração de alguns parâmetros é trabalhosa e pode influenciar significativamente no desempenho do algoritmo, fazendo com que abordagens mais simples às vezes apresentem o melhor resultado.

Outra conclusão tirada a partir dos resultados computacionais foi que o custo-benefício entre tempo e qualidade em problemas difíceis deve ser levado em conta ao se trabalhar com heurísticas. Isso foi percebido ao comparar os resultados do GRASP, quando o aumento de quatro vezes no tempo de busca levou a um pequeno ganho na solução.

Referências

[1] M. Ahmadvand1, M.Yousefikhoshbakht and N. M. Daran, "Solving the Travelling Salesman Problem by an Efficient Hybrid Metaheuristicn Algoritm."Journal of Advances in Computer Research vol.3, No.3, August 2012, pages 75-84.

[2] Arenales M., Armentano V., Morabito R., Yanasse H., "Pesquisa Operacional."Elsevier Editora Ltda, 2007.

[3] Feo T.A. and Resende M.G.C.(1995) "*Greedy Randomized Adaptive Search Procedures.*"Journal of Global Optimization,6:109-133.

[4] Oliveira J. F., Carravilla M. A., "Metodologia de Apoio à Decisão."MEEC-FEUP.1998.

[5] Gomes, F.A.M., "Busca Local Iterada."Unicamp.

[6] M. Gendreau, J.-Y. Potvin (eds), Handbook of Metaheuristics, Springer, 2nd ed., 2010.

[7] Freddo A. R.and Brito R. C., "Implementação da Metaheurística GRASP para o Problema do Caixeiro Viajante Simétrico."Univerisade Federal do Paraná.

[8] "Algoritmo de Busca Local"Pontifícia Universidade Católica de Minas Gerais. Certificação Digital N.0210675/CA.

[9] T.A.Feo and M.G.C. Resende(1989)"A probabilistic heuristic for a computationally dificult set covering problem."*Operations Research Letters*, 8:67-71,1989.