Abordagem baseada em GRASP para o *Travelling Thief Problem*

Daniel K. S. Vieira

29 de junho de 2016

Resumo

Problemas do mundo real normalmente compostos por componentes interdependentes. Neste caso, problemas de referência que não representam esta interdependência não são uma boa escolha para avaliar o desempenho de algorítimos. Na literatura recente, um problema de referência chamado de *Travelling Thief Problem* (TTP) foi proposto para representar de forma mais adequada problemas multicomponente do mundo real. TTP é uma combinação de dois problemas de referência muito conhecidos na computação: Problema da Mochila Inteira ou 0-1 *Knapsack Problem* (KP) e o Problema do Caixeiro Viajante ou *Travelling Salesman Problem* (TSP).

Este artigo propõe uma abordagem baseada em *Greedy Randomized Adaptative Search Procedures* (GRASP) [5] chamada de *Repetitive Greedy with Local Search* (RGLS) feita para obter soluções competitivas nas mais variadas instâncias do TTP. esta heurística foi testada em 72 instâncias representativas e comparadas com um Algoritmo Genético (AG) chamado de *Multi-component Genetic Algorithm* (MCGA) proposto por Vieira e Mendes [11]. Os experimentos mostram que o RGLS obteve um desempenho superior ao MCGA em quase todas as instâncias testadas.

1 Introdução

Problemas clássicos da computação foram propostos e atualmente são estudados para que se definam estratégias que obtenham uma boa solução de forma eficiente para problemas do mundo real (em que estes problemas clássicos são inspirados). Muitos desses problemas pertencem a classe NP-difícil, assim, não é possível encontrar a solução ótima em tempo polinomial. Dessa forma, a utilização de meta-heurísticas é salutar, uma vez que pode encontrar boas soluções em um tempo aceitável.

De acordo com Bonyadi et al. [3], problemas do mundo real geralmente possuem componentes interdependentes e estes não devem ser resolvidos separadamente. Esta correlação deve ser levada em consideração para que se possa obter uma solução adequada para o problema como um todo.

Com o objetivo de melhor abranger a complexidade de um problema do mundo real, um novo problema de referência foi proposto chamado de Travelling Thief Problem (TTP) [3], que é uma junção de dois problemas muito conhecidos na computação que são o Knapsack Problem (KP) e o Travelling Salesman Problem (TSP).

Várias estratégias foram propostas com o objetivo de obter soluções satisfatórias no TTP de forma eficiente. Faulkner et al. [4] apresentam uma série de algoritmos que focam em manipular a componente KP do problema com a componente TSP sendo obtida pela heurística Chained Lin-Kernighan (CLK) [1]. Bonyadi et al. [2] utilizam uma abordagem co-evolucionária chamada de CoSolver em que módulos distintos são responsáveis pelos componentes do TTP e estes se comunicam e combinam soluções para obter uma solução geral aproximada do problema. Desta maneira o CoSolver tenta resolver o problema TTP trabalhando com as duas componentes (KP e TSP) simultaneamente e não sequencialmente como em [4]. Mei et al. [8] focam em instâncias de larga escala propondo estratégias de redução de complexidade para o TTP com esquemas de aproximação de fitness e aplica estas técnicas em um Algoritmo Memético (AM) que supera Busca Local Aleatória e o Algoritmo Evolucionário proposto por Polyakovskiy et al. [10]. Oliveira et al. [9] desenvolveram uma abordagem baseada em Busca-Tabu (BT) com uma busca local chamado 2-OPT que se mostrou ter um desempenho competitivo para instâncias de tamanho reduzido. Vieira e Mendes [11] propuseram um AG chamado de *Multi-component Genetic Algorithm* (MCGA) que obteve resultados notáveis em instâncias com até 783 cidades.

Este artigo propõe uma heurística baseada em GRASP [5] chamada de *Repetitive Greedy with Local Search* (RGLS) em que a solução inicial é criada utilizando CLK para a componente TSP do problema e uma heurística gulosa para a componente KP mas com os valores dos itens modificados em função da sua distância do final da rota, ou seja, quanto mais perto do início da rota um item está, menos ele vale. Após gerar a solução inicial um algoritmo conhecido como *Bit-flip* é utilizada para fazer uma busca local na solução corrente. Caso após várias várias tentativas a solução não tenha melhorado ela é armazenada caso seja a melhor encontrada até o momento e o processo começa novamente.

O artigo é divido da seguinte forma: a seção 2 descreve o TTP; a análise de complexidade do MCGA é desenvolvida na seção 3; na seção 4 a heurística proposta neste trabalho é descrita e analisada; os testes e comparações realizadas entre o RGLS e o MCGA são apresentados na seção 5; e finalmente na seção 6 são expostas as conclusões obtidas e ideias para trabalhos futuros.

2 Travelling Thief Problem

Com base em Polyakovskiy et al. [10], o TTP é definido como tendo um conjunto de cidades $N = \{1, \ldots, n\}$ em que a distância d_{ij} entre cada par de cidades i e j é conhecida, com $i, j \in N$. Toda a cidade i exceto a primeira possui um conjunto de itens $M_i = \{1, \ldots, m_i\}$. Cada item k existente em uma cidade i é

caracterizado pelo seu valor p_{ik} e peso w_{ik}. A solução candidata deve visitar todas as cidades sem repetir nenhuma e retornar para a cidade de partida.

Itens podem ser coletados pelas cidades respeitando a restrição de capacidade máxima W da mochila. Uma taxa de aluguel R deve ser paga por cada unidade de tempo que se utiliza para finalizar o percurso. υ_{max} e υ_{min} descrevem a velocidade máxima e mínima permitida ao longo do percurso, respectivamente.

Seja $y_{ik} \in \{0, 1\}$ uma variável binária igual a 1 caso o item k fosse coletado na cidade i. Sendo W_i o peso total dos itens coletados quando se deixa a cidade i. Portanto, a função objetivo para um percurso $\Pi = (x_1, \ldots, x_n)$, $x_i \in N$ e um plano de coleta $P = (y_{21}, \ldots, y_{nm_i})$ é definida como:

$$\begin{aligned} Z(\Pi, P) = & \sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{ik} y_{ik} \\ & - R \left(\frac{d_{x_n x_1}}{\upsilon_{max} - \nu W_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{\upsilon_{max} - \nu W_{x_i}} \right) \end{aligned} \tag{1}$$

em que $\nu = \frac{\upsilon_{max} - \upsilon_{min}}{W}$. O objetivo é **maximizar** $Z(\Pi, P)$. A equação 1 se resume em penalizar os lucros obtidos pelos itens coletados com um valor que representa o tempo total que se leva para completar o percurso Π multiplicado pela taxa de aluguel R.

A figura 1 mostra um exemplo do TTP no qual se tem o mesmo número de itens para cada cidade assim como nas instâncias de teste fornecidas por Polyakovskiy et al. [10], Neste caso tem-se 2 itens para cada cidade (exceto a primeira cidade, que não possui itens) e 3 cidades ao todo. Cada item $I_{ij}(p_{ij}, w_{ij})$ é descrito como sendo o item j da cidade i que possui um valor p_{ij} e um peso w_{ij}. Assumindo a capacidade máxima da mochila como sendo $W = 10$, a taxa de aluguel $R = 1$, $\upsilon_{max} = 1$ e $\upsilon_{min} = 0.1$. Logo, uma solução possível é $P = (0, 1, 0, 1)$ e $\Pi = (1, 2, 3)$, descrevendo que serão coletados os itens I_{22}, I_{32} que estão nas cidades 2 e 3, respectivamente, fazendo uma rota que partindo da cidade 1 vai para a cidade 2, depois para a 3 e por fim retornando à cidade 1. Isso resulta em $Z(\Pi, P) \approx -28.053$.

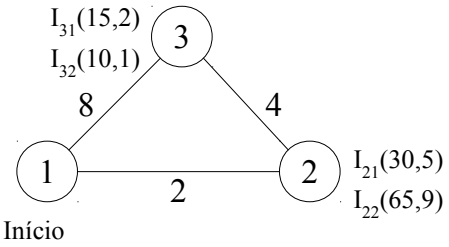


Figura 1: Exemplo de instância do TTP.

Note que neste exemplo, caso apenas o componente PCV do problema for considerada, todas as possibilidades possuem custos iguais já que todas as co-

nexões entre as cidades serão utilizadas de qualquer forma. Mas caso leve-se em consideração o problema como um todo, é possível notar que a ordem da rota percorrida passa a influenciar no custo da solução graças a variação de tempo que um item é mantido na mochila. Em outras palavras, um item pesado coletado no começo do percurso influencia a velocidade por um período de tempo maior se comparado com o mesmo item coletado no final do percurso, tornando a solução mais lenta e aumentando o custo.

3 Análise do MCGA

O AG proposto por Vieira e Mendes [11] aplica um tipo de operador distinto para cada componente do problema. São estes:

- Cruzamento: *N-point* no componente KP e o *Order-based* no componente TSP;
- Mutação: *Bit-flip* no componente KP e 2-OPT no componente TSP.

Para a etapa de seleção o operador chamado de Torneio foi utilizado. A Figura 2 mostra um fluxograma descrevendo as etapas de execução do MCGA.

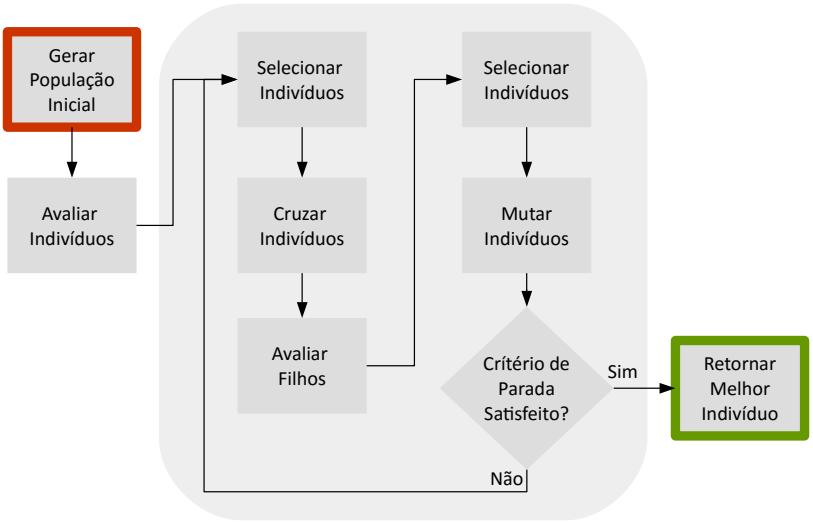


Figura 2: Fluxograma de execução do MCGA.

A seguir será feita a análise de complexidade temporal destes operadores juntamente com a função de avaliação para se concluir qual a complexidade deste GA levando em conta a execução de apenas uma geração.

O operador *N-point* combina os cromossomos dos pais alternando entre seus genes na geração dos filhos. Isso significa que deve-se iterar por todos os genes

do cromossomo do componente KP e isso possui um custo de $O(m)$, em que m representa o número total de itens da instância.

O *Order-based* que é aplicado no componente TSP combina as rotas de tal forma que não se gera indivíduos infactíveis, logo, não é necessário fazer uma verificação dos filhos gerados. A complexidade deste operador é definida pela criação da máscara, ordenações e atribuições feitas, resultando em um custo computacional de $O(n\log(n)+n) = O(n\log(n))$, sendo n o número de cidades.

Calculando um valor randômico por iteração, iterando sobre todo o cromossomo do KP, o *Bit-flip* possui uma complexidade de $O(m)$. Mas este operador pode gerar indivíduos infactíveis ultrapassando a capacidade da mochila. Quando isso ocorre é removido os itens com menor razão de valor por peso até que o indivíduo se torne factível. O custo desta operação é definido pelo cálculo das razões ($O(m)$), a ordenação dos itens coletados ($O(m\log(m))$) e por fim a remoção dos itens necessários verificando se o indivíduo se tornou factível a cada remoção ($O(m)$), resultando em uma complexidade de $O(m\log(m)+2m) = O(m\log(m))$.

Já o operador 2-OPT pode ser realizado invertendo uma parte do cromossomo do TSP, logo este tem um custo computacional de $O(n)$.

A etapa de seleção é feita sobre os pais e filhos resultando no dobro de operações (número de filhos gerados é iguam a de pais existentes) mas sendo uma constante a classe de complexidade se mantem. O custo da operação de torneio varia em função do número de indivíduos que se deseja como resultado e o tamanho do torneio. Definindo s como sendo o número de indivíduos que devem ser retornados e k o tamanho do torneio, tem-se que a complexidade deste operador é $O(s \times k)$.

Este AG trabalho com o conceito de elitismo e portanto precisa resguardar b melhores indivíduos de operações que possam os eliminar. Isto pode ser feito realizando uma ordenação parcial sobre os indivíduos. Utilizando um estrutura *max-heap* se insere os b primeiro elementos e depois se inseri os restantes um de cada vez removendo o maior elemento após cada inserção. Como cada inserção possui um custo de $O(\log(b))$, o custo para i indivíduos é $O(i\log(b))$.

A avaliação de um indivíduo consiste em computar a equação 1. Portanto o que se faz é calcular o valor total obtido pelos itens coletados penalizado pelo tempo para se percorrer o trajeto, coletando os itens a medida que se realiza a rota definida. Isto pode ser calculado com um custo de $O(m+n)$ [8]. O fato da avaliação ser feita duas vezes no MCGA não altera sua classe de complexidade.

A população inicial é formada por indivíduos com a mochila vazia e a rota definida pela heurística CLK que basicamente aplica o algoritmo de busca local *Lin-Kernighan* (LK) [7], gera uma perturbação na solução obtida e aplica o algoritmo novamente. Esse processo se repete até que alguma condição de parada seja satisfeita (pode ser tempo de execução, qualidade da solução, etc.). Como esta heurística não interrompe sua execução em um número bem definido de instruções executadas, apenas uma iteração do LK será levada em consideração na análise de complexidade. De acordo com Helsgaun [6], LK pode ser implementado com uma complexidade de aproximadamente $O(n^{2.2})$.

Portanto a complexidade geral do MCGA em uma geração para um indivíduo

é:

$$O(m + n\log(n) + m + m\log(m) + n + (s \times k) + i\log(b) + (m + n) + (n^{2.2})) \tag{2a}$$

$$= O(n^{2.2} + sk + m\log(m) + i\log(b)) \tag{2b}$$

Mas esta complexidade é referente a apenas um indivíduo (exceto o custo gerado pelo Torneio e elitismo que são referentes à geração como um todo). Tendo um número i de indivíduos e sabendo que o Torneio foi configurado para retornar o mesmo tamanho da população de entrada, tem-se que a complexidade do MCGA para uma geração é de:

$$O(i(n^{2.2} + k + m\log(m) + \log(b))) \tag{3}$$

4 *Repetitive Greedy with Local Search*

A meta-heurística GRASP [5] especifica que soluções são obtidas a partir de um processo repetitivo de construção de uma solução viável a partir e uma heurística gulosa randômica e posteriormente é feito uma busca local nestas soluções. A melhor solução obtida é retornada como final.

Como já foi dito anteriormente, a heurística RGLS proposta neste trabalho se baseia em GRASP, e como tal, possui dois estágios principais: construção da solução inicial e busca local.

Na etapa de construção da solução inicial o CLK é utilizado para obter uma solução viável para o componente TSP e como o CLK já contempla uma busca local a rota obtida não é alterada posteriormente. Os valores dos itens são modificados em função da distância que estão do final da rota construída pelo CLK e uma heurística gulosa é aplicada sobre o componente KP sobre estes itens modificados.

A etapa de busca local consiste em aplicar o operador *Bit-flip* (herdado do MCGA e descrito posteriormente) e caso a solução tenha melhorado com a aplicação do operador, a nova solução é armazenada e o operador é aplicado novamente. Esta busca local é interrompida quando alguma condição de parada é satisfeita. As condições de parada utilizadas serão descritas na seção 5. Quando a busca local é finalizada a solução obtida é armazenada e o processo se inicia novamente.

A Figura 3 apresenta o fluxograma de execução do RGLS.

Para alcançar o nível de diversidade de soluções iniciais adequada para o GRASP, as soluções retornadas pelo CLK são obtidas alternando de forma randômica entre tipos de perturbação e soluções iniciais que ele utiliza.

As soluções inicias são geradas utilizando estas heurísticas: Randômico; *Nearest-Neighbor*; Guloso; *Quick-Borůvka*. As perturbações (*kick*) foram realizadas utilizando os seguintes algoritmos: Randômico; Geométrico; *Random-walk*; *Close*. Todos os algoritmos e heurísticas são contemplados no trabalho de Applegate et al. [1].

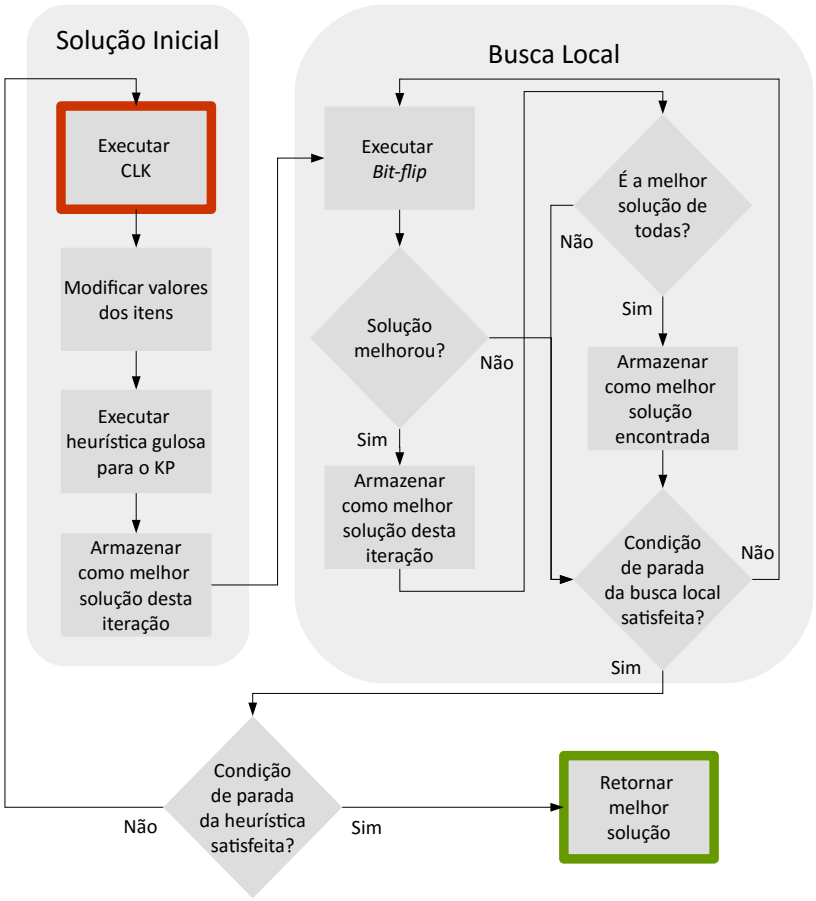


Figura 3: Fluxograma de execução do RGLS.

A modificação dos valores dos itens é feita da seguinte forma: dado um item i com valor p_i, seu novo valor p'_i será

$$p'_i = p_i + \frac{p_i}{\sqrt{d_i}}, \tag{4}$$

sendo d_i a distância percorrida para alcançar o item i. Ou seja, quanto mais perto um item está da origem da rota, mais seu valor aumenta. A raiz quadrada da distância foi utilizada para tentar diminuir o impacto que a distância causa no novo valor do item. Não utilizar a raiz quadrada faz com que a segunda componente da Equação 4 fique muito próximo de 0 rapidamente e, consequentemente, muitos itens não tem seu valor alterado significativamente.

Para selecionar os itens iniciais que estarão na mochila uma abordagem gulosa foi utilizada. Nela é definido uma pontuação s_i para cada item i como descrito a seguir:

$$s_i = \frac{p'_i}{w_i} \tag{5}$$

Os itens com maiores pontuações são coletados enquanto não se exceda a capacidade da mochila.

O operador *Bit-flip* é um algoritmo muito simples que é utilizado para alternar itens coletados de forma randômica. Os itens são coletados caso não estejam na mochila ou retirados caso contrário com uma probabilidade α (normalmente muito pequena). Como a adição de um item utilizando este operador pode acarretar na infactibilidade da solução, a função de avaliação penaliza criticamente tal solução definindo seu desempenho como $-\infty$ para que ela seja descartada posteriormente.

4.1 Análise de complexidade do RGLS

A solução para o componente TSP é obtida através do CLK que como já foi mencionado anteriormente possui complexidade $O(n^{2.2})$. A modificação dos valores dos itens e o *Bit-flip* são claramente pertencente a classe $O(m)$ já que obrigatoriamente se passa por todos os itens.

Já que a escolha dos itens possui custo linear caso os itens já estejam ordenados com base em suas pontuações, a heurística gulosa aplica sobre o componente KP possui complexidade de $O(m \log(m))$.

A avaliação da solução eleva o custo em $O(m + n)$ como já foi citado na seção 3. Mesmo sendo feito duas avaliações na primeira iteração (uma etapa de solução inicial e a outra em cada iteração da etapa de busca local) a análise de complexidade se mantém.

Note que para alcançar essas classes de complexidade o RGLS foi implementado com a rota e o plano de coleta sendo armazenados de forma contígua em memória, possibilitando acesso aos valores em $O(1)$.

Portanto a complexidade temporal geral do RGLS para a primeira iteração é:

$$O(n^{2.2} + 2m + (m + n) + m \log(m)) \tag{6a}$$

$$= O(n^{2.2} + m \log(m)) \tag{6b}$$

5 Metodologia e Resultados

Um subconjunto representativo das 9720 instâncias de teste [10] foi utilizado para comparar o desempenho do MCGA e do RGLS. Estas instâncias propostas por Polyakovskiy et al. possuem de 51 à 85.900 cidades n, número de itens por cidade $F \in \{1, 3, 5, 10\}$ totalizando m itens, 10 níveis de capacidade da mochila C, e três tipos de Problema da Mochila t: *uncorrelated* (u – o valor do item não é correlacionado ao seu peso), *uncorrelated with similar weights* (usw – mesmo que o *uncorrelated* mas os itens possuem pesos similares) e *bounded strongly*

correlated (*bsc* – itens com valores altamente correlacionados aos seus pesos e é possível existir itens com as mesmas características).

Configurações distintas foram definidas para o RGLS e MCGA com o objetivo de fazer com que estas heurísticas se adaptem aos variados tamanhos de instâncias testadas. As configurações definidas para o MCGA são:

M1:
- Nº de indivíduos: 200
- Tamanho do torneio: 2
- Nº de soluções de elite: 12
- Nº de pontos para o *N-point*: 3
- Probabilidade α para o *Bit-flip*: 0.2%
- Condição de parada: 10 minutos de execução
- Tempo de execução do CLK: Não há restrição

M2:
- Nº de indivíduos: 80
- Tamanho do torneio: 2
- Nº de soluções de elite: 6
- Nº de pontos para o *N-point*: 3
- Probabilidade α para o *Bit-flip*: 0.2%
- Condição de parada: 50 minutos minutos de execução
- Tempo de execução do CLK: $\frac{0.6t_{MCGA}}{i}$, em que t_{MCGA} é o tempo de execução que o MCGA como um todo possui e i é o tamanho da população

Note que o tempo de execução do CLK também é levado em consideração no momento de contabilizar o tempo de execução das heurísticas. Já o RGLS possui as seguintes configurações:

R1:
- Probabilidade α para o *Bit-flip*: 0.1%
- Condição de parada para a etapa de busca local: 10000 iterações sem melhoria
- Condição de parada da heurística: 10 minutos de execução

R2:
- Probabilidade α para o *Bit-flip*: 0.1%
- Condição de parada para a etapa de busca local: 10000 iterações sem melhoria
- Condição de parada da heurística: 50 minutos de execução

Os testes foram realizados em um servidor com dois Intel Xeon E5-2630 v3 totalizando 32 núcleos, 128 GB de RAM e sistema operacional Ubuntu 14.04 x86-64 LTS.

O subconjunto de instâncias testadas foi dividido em um conjunto com instâncias menores (com o número de cidades variando de 195 até 3038) em

que as configurações M1 e R1 foram utilizadas para as heurísticas, e outro conjunto com instâncias maiores (11849 à 85900 cidades) no qual M2 e R2 foram as configurações utilizadas. Os resultados dos testes são apresentadas nas Figuras 4 e 5, respectivamente.

Para as instâncias menores 10 execuções de cada heurística foram feitas e a média dos resultados forma normalizados de 0 à 1 utilizando o menor e maior valor encontrado pelas duas heurísticas como extremos. Para as instâncias maiores 5 execuções foram realizadas.

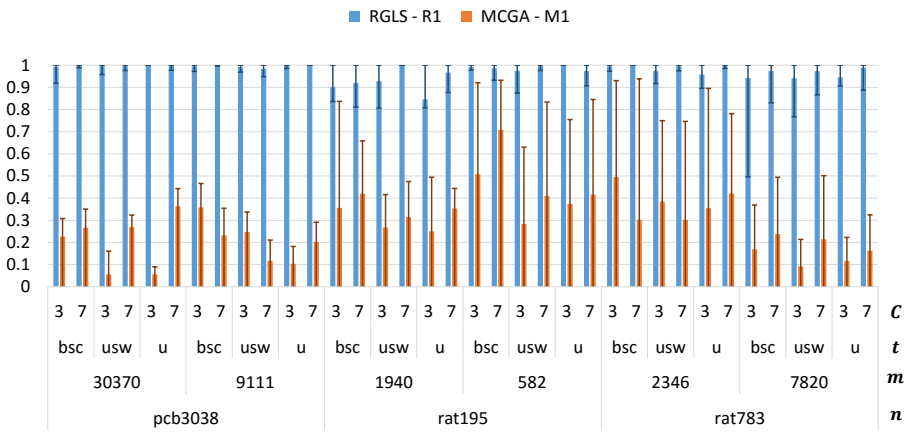


Figura 4: Resultados obtidos nas instâncias de 195 à 3038 cidades.

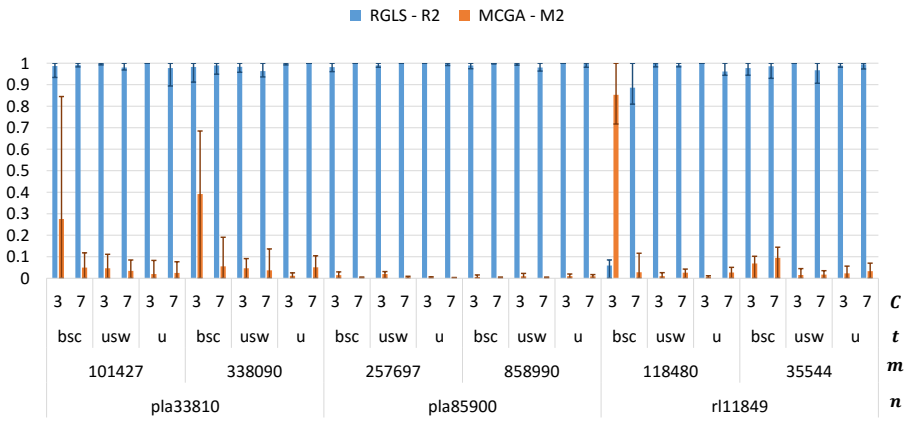
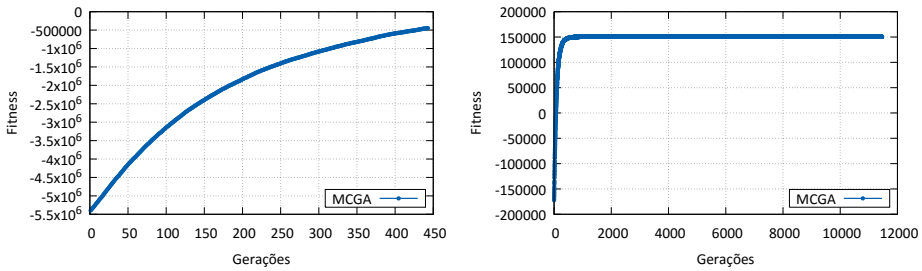


Figura 5: Resultados obtidos nas instâncias de 11849 à 85900 cidades.

É notável a superioridade do RGLS principalmente nas instâncias maiores como já era previsto de acordo com a análise de complexidade demonstrada e testes preliminares realizados utilizando o MCGA com uma configuração similar

a M1 e apresentados na Figura 6. Nestes testes é possível observar o comportamento de convergência para uma instância menor (com 100 cidades) o que não se repete em um instância maior (com 3038 cidades), comprovando a ineficiência do MCGA em instâncias maiores.

O desempenho do RGLS não é só superior mas é mais estável também, obtendo uma média quase sempre bem próxima do melhor resultado encontrado pelas heurísticas e com uma variação (mostrada pelas barras de erro que indicam a variação entre o pior e melhor resultado encontrado) relativamente pequena na maioria das instâncias se comparado com o MCGA.



(a) Instância `pcb3038_n30370_uncorr_07`. (b) Instância `kroA100_n990_uncorr_10`.

Figura 6: Evolução do MCGA.

É interessante observar a instância com 11849 cidades, 118480 itens, capacidade da mochila de categoria 3 e KP do tipo *bounded strongly correlated*. Nesta instância o MCGA obteve um desempenho superior, obtendo uma média mais próxima do melhor resultado encontrado pelas duas heurísticas enquanto o RGLS obteve o comportamento inverso.

Suspeitava-se que alguma carga de trabalho tivesse sobrecarregado o servidor impactando a execução do RGLS mas uma investigação posterior foi feita e observou-se que em média 81 iterações completas (desde a obtenção da solução inicial) foram processadas, logo, falta de tempo de processamento não foi a causa deste desempenho negativo por parte do RGLS.

6 Conclusões

Problemas heterogêneos composto por vários outros subproblemas se mostram bastante desafiadores por conta das consequências ainda não profundamente estudadas da interdependência entre estes subproblemas.

Neste trabalho uma abordagem baseada em GRASP chamada de *Repetitive Greedy with Local Search* (RGLS) foi proposta com o objetivo de resolver um problema combinatório multicomponente chamado de *Travelling Thief Problem*. Esta heurística obtém a solução inicial utilizando um algoritmo baseado em busca para uma componente do problema e um algoritmo guloso para o outro componente. Após isto, uma busca local é realizada sobre uma das componentes

e todo o processo começa novamente.

Os resultados dos testes realizados mostram que o RGLS obtém soluções de alta qualidade em instâncias variando de 195 à 3038 cidades e satisfatórias em instâncias maiores (com um número de cidades variando de 11849 até 85900), bem superiores aos de uma abordagem baseada em Algoritmo Genético conhecida como MCGA, Esta superioridade que o RGLS possui sobre o MCGA é, em grande parte, devido a sua complexidade temporal ligeiramente inferior, possibilitando processar instâncias maiores em menos tempo.

Um estudo mais aprofundado deve ser realizado para verificar o motivo pelo desempenho peculiar que o RGLS obteve em uma das instâncias testadas (podendo haver mais). Estratégias melhores para busca local que atuem sobre os dois componentes do problema também devem ser pesquisadas na tentativa melhorar da qualidade das soluções. A alteração do algoritmo guloso por uma heurística mais randômica também pode ajudar a obter soluções iniciais com maior diversidade, característica recomendada para uma estratégia baseada em GRASP.

Referências

[1] David Applegate, William Cook, and André Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.

[2] Mohammad Reza Bonyadi, Zbigniew Michalewicz, Michaŏ Roman Przybyŏek, and Adam Wierzbicki. Socially inspired algorithms for the travelling thief problem. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 421–428. ACM, 2014.

[3] M.R. Bonyadi, Z. Michalewicz, and L. Barone. The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 1037–1044, June 2013.

[4] Hayden Faulkner, Sergey Polyakovskiy, Tom Schultz, and Markus Wagner. Approximate approaches to the traveling thief problem. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pages 385–392. ACM, 2015.

[5] Thomas A. Feo and Mauricio G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[6] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106 – 130, 2000.

[7] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[8] Yi Mei, Xiaodong Li, and Xin Yao. Improving efficiency of heuristics for the large scale traveling thief problem. In *Simulated Evolution and Learning*, pages 631–643. Springer, 2014.

[9] Matheus RR Oliveira, André Gustavo dos Santos, and Matheus de Freitas Araujo. Uma heurística busca tabu para o problema do mochileiro viajante. In *Simpósio Brasileiro de Pesquisa Operacional*, SBPO '15, 2015.

[10] Sergey Polyakovskiy, Mohammad Reza Bonyadi, Markus Wagner, Zbigniew Michalewicz, and Frank Neumann. A comprehensive benchmark set and heuristics for the traveling thief problem. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 477–484, New York, NY, USA, 2014. ACM.

[11] Daniel Vieira and Marcus Mendes. A genetic algorithm for multi-component optimization problems: the case of the travelling thief problem. 2016.