*Artigo aceito para o XXXVII SBPO 2005 – Gramado/RS*

# DESENVOLVIMENTO E ANÁLISE EXPERIMENTAL DA HEURÍSTICA GRASP APLICADA A UM PROBLEMA DE COLETA SELETIVA

**Márcia Cristina Valle Zanetti**
Instituto de Computação – Programa de Pós Graduação em Computação
Universidade Federal Fluminense e
Centro de Ensino Superior de Juiz de Fora - CES/JF

**Luiz Satoru Ochi**
Instituto de Computação – Programa de Pós Graduação em Computação
Universidade Federal Fluminense
Rua Passo da Pátria, 156, 3o andar, Niterói – RJ
satoru@ic.uff.br

**Resumo**

Este artigo apresenta diferentes algoritmos heurísticos de construção e busca local para o *Greedy Randomized Adaptive Search Procedures* (GRASP) para resolver uma generalização do Problema do Caixeiro Viajante conhecido na literatura como *The Traveling Purchaser Problem* (TPP). O TPP consiste em determinar uma rota de um comprador para este adquirir um conjunto de produtos em diferentes mercados de forma a minimizar o custo total de aquisição dos produtos e de percurso. Extensivos experimentos computacionais mostram o desempenho de cada versão proposta. Os resultados computacionais mostram que algumas das versões propostas são robustas tanto em relação à adaptabilidade às características das instâncias analisadas como também na rapidez em convergir para soluções sub-ótimas.

**Palavras-chave**: metaheuristica, GRASP, problema do caixeiro viajante, problema de coleta seletiva.

**Abstract**

This paper presents several heuristic algorithms for construction and local search phases of Greedy Randomized Adaptive Search (GRASP) to solve a generalization of the Traveling Salesman Problem called in the literature as The Traveling Purchaser Problem (TPP). The TPP is the problem of determining a tour of a purchaser buying a set of items in different shops by minimizing the total amount of travel and purchase costs. Extensive computational experiments show the performance of each proposed version. The results obtained indicate that some proposed algorithms are robust and provide good solutions for moderate size instances in just a few seconds.

**Keywords**: metaheuristic, GRASP, traveling salesman problem, traveling purchaser problem.

## 1. Introdução

Este trabalho aborda a solução de um Problema de Coleta Seletiva (PCS) denominado *The Traveling Purchaser Problem* (TPP) [4, 9, 15]. O problema é descrito da seguinte forma: Existem um conjunto de *n* produtos a serem adquiridos. Para isso estão disponíveis *m* mercados. Cada produto se encontra disponível em pelo menos um mercado e o preço de cada produto pode variar de mercado a mercado. O objetivo do PCS, é adquirir estes *n* produtos visitando um subconjunto dos *m* mercados de modo que os custos de aquisição e de percorrer

a rota associada sejam minimizados. Uma solução deste problema pode ser representada por uma rota iniciando e finalizando num local de origem, passando por um subconjunto de mercados e de forma a atender as restrições do problema que incluem as restrições do Problema do Caixeiro Viajante (PCV) além da exigência de compra dos *m* produtos a um menor custo total possível. São dados do problema:

- Conjunto M de *m* mercados acrescidos da origem $m_o$, onde na origem não existem produtos disponíveis. M = $\{m_0; m_1; m_2; ...; m_m\}$.
- Conjunto K de *n* produtos a serem adquiridos em M, K = $\{k_1; k_2; k_3; ...; k_n\}$.
- Matriz T = $(t_{ij})$: $0 \le i, j \le m$, onde $t_{ij}$ representa o custo de se locomover do mercado i para j.
- Matriz P = $(p_{kj})$: $1 \le k \le n$, $1 \le j \le m$, onde $p_{kj}$ representa o custo do produto *k* no mercado *j*. Na indisponibilidade de um produto *k* num mercado *j*, será dado um custo muito elevado como penalidade (por ex. a *soma dos custos de todos os produtos em todos os mercados*).

Embora o PCS como definido neste trabalho seja aplicável em muitos problemas reais de roteamento e *scheduling*, a literatura não apresenta muitas contribuições sobre ele. O PCS ou o TPP é uma generalização do *Problema do Caixeiro Viajante (PCV)* e foi introduzido por Ramesh em 1981 [9]. O caso particular onde o PCS se torna o PCV, é quando $m = n$, e cada mercado (vértice) possui disponível um único tipo de produto distinto dos demais mercados. Neste caso, a rota do PCS necessariamente deve passar por todos os vértices do grafo associado. Portanto o PCS é também enquadrado como um problema NP-Completo.

Devido ao fato do PCS ser computacionalmente intratável, algumas heurísticas foram propostas na tentativa de resolvê-lo de forma aproximada. Dentre elas podemos citar: Ramesh [9] apresentou uma heurística convencional e uma busca lexicográfica para o PCS. Em [4] os autores estendem os resultados de [9] e apresentam uma heurística para o PCS usando a noção de inserção e economias. Heurísticas de inserção e descarte de vértices do tipo ADD e DROP são vistos em [4, 6, 7, 8, 10, 12, 14, 15]. Metaheuristicas baseadas em GRASP e VNS são abordadas em [12, 14], mas a ênfase nos dois trabalhos é a apresentação de modelos paralelos e distribuídos. Pouca importância é dada ao desenvolvimento de algoritmos seqüenciais. Uma busca tabu é apresentada em [15] onde o autor apresenta uma nova idéia de gerenciar listas tabu de forma dinâmica. Os resultados são expressivos mas como todos os trabalhos anteriores, este também não disponibiliza as instâncias e os resultados obtidos numa biblioteca pública. Os autores de [12, 14] no entanto implementaram a Busca Tabu proposta em [15] e numa comparação com o GRASP proposto, mostraram que ambos apresentam desempenho muito similar tanto em relação à qualidade das soluções geradas como no tempo computacional exigido. Este resultado nos motivou a desenvolver versões GRASP mais sofisticados na solução deste problema.

Este trabalho tem como objetivo, desenvolver e analisar o desempenho de heurísticas GRASP aplicadas na solução do PCS. Para isso são propostos diferentes métodos de construção que possuem as seguintes características: são randomizados, adaptativos, gulosos e iterativos bem como diferentes formas de busca local. Adicionalmente é proposto para o PCS, um método de busca intensiva conhecido na literatura como "*Path Relinking*" (Reconexão de Caminhos) [1, 3].

Resultados computacionais efetuados mostram o potencial destas heurísticas na solução de diferentes problemas testes do PCS. Numa etapa seguinte, é proposto uma análise probabilística das heurísticas propostas com o intuito de analisar a robustez dos mesmos. Resultados de diferentes simulações mostram um futuro promissor para estas metodologias na solução de problemas de elevada complexidade computacional como é o caso do PCS. Este artigo apresenta nas próximas seções: Descrição dos algoritmos de construção, busca local e reconexão de caminhos na seção 2, a seção 3 apresenta os resultados computacionais obtidos e na seção 4, são mostrados as conclusões e propostas de trabalhos futuros.

## 2. Algoritmos Propostos

O objetivo deste artigo, é tanto desenvolver como analisar o desempenho de diferentes versões da heurística GRASP aplicados ao PCS. Para tanto são propostos vários métodos de construção e busca local adaptados a estrutura do GRASP. Ao colocar esta meta, queremos tirar informações sobre o impacto de cada método de construção e busca local usado no desempenho final do GRASP. Incluímos também um estudo sobre a viabilidade de usar buscas locais adicionais no GRASP, sem onerar significativamente o seu tempo final. Para tanto, é proposto um módulo de reconexão de caminhos (RC) proposto inicialmente por Glover para a metaheuristica Busca Tabu [3].

### 2.1 Algoritmos de Construção Randomizada

Este trabalho apresenta quatro algoritmos, com características distintas, para a construção de soluções iniciais viáveis. As diversas versões propostas neste trabalho permitem avaliar quais as características que melhor se adaptam ao PCS.

#### 2.1.1 ADD Randomizado (ADDR)

O algoritmo **ADD básico** [8, 10] constitui-se no critério guloso de inserção de vértices. A solução parcial do PCS, que inicialmente é caracterizada somente pela origem, será construída passo a passo, através do seguinte critério de inserção: a partir de uma lista de candidatos, contendo todos os vértices não pertencentes à solução parcial, deverá ser selecionado um vértice *k*, que será o próximo a ser inserido na solução. O critério de seleção do vértice *k* é baseado no conceito de economia, isto é, a cada passo, será escolhido o vértice que ofereça a *melhor economia*, caso seja inserido na solução parcial. Os vértices são inseridos um a um na solução parcial até que uma solução viável do PCS possa ser alcançada.

Neste trabalho o algoritmo ADD foi adaptado para se adequar à filosofia do GRASP**,** resultando numa variação que chamamos de ADD *randomizado* (ADDR). Essa variação consiste na presença do componente probabilístico do GRASP, que é associado ao algoritmo ADD. Para tal, antes da inserção de um novo vértice à solução, é gerada uma Lista de Candidatos Restritos (LCR) composto por um percentual $\alpha$ dos melhores candidatos , e em seguida, um dos candidatos de LCR é selecionado aleatoriamente. Obviamente se o percentual considerar apenas um candidato, ele será o próprio guloso (ADD), e se o percentual considerar os 100% dos candidatos, a escolha será totalmente aleatória. Esta seleção aleatória num subconjunto de candidatos permite que sempre um bom candidato seja escolhido (dependendo obviamente do tamanho da LCR) e ao mesmo tempo permite que soluções distintas sejam obtidas a cada execução do mesmo.

Para o PCS, o passo de inserção será em relação à escolha de um novo mercado a ser inserido na solução parcial. Para ordenar os mercados a cada passo da construção, o seguinte procedimento é adotado: para cada mercado ainda fora da solução corrente, *calcula-se o preço médio dos produtos que ainda não foram adquiridos na Lista de Compras em cada mercado.* Essa média consiste na soma do preço de todos os produtos ainda não adquiridos na solução parcial, incluindo o custo dos produtos que o mercado não oferece, que, nesse caso, recebe um valor elevado, dividida pelo total de produtos ainda a serem comprados. Essa MÉDIA será um indicador de eficiência do mercado no momento atual. Só então, de posse das médias de cada mercado, será calculado o CUSTO da inserção desse mercado na solução. O cálculo do CUSTO consiste em adicionar, à média de preços dos produtos mais o custo de inserir mais este mercado na solução atual. Logo, partindo-se de uma solução parcial, que contém somente o ponto de origem, das médias de preços de cada mercado e da distância entre os diversos mercados, pode-se, então, calcular o custo de inserção de cada mercado j na solução parcial:

A economia associada à inserção um mercado na solução parcial será obtida através da diferença em valores absolutos entre o *custo da solução antes da inserção e o custo após o acréscimo de um novo mercado candidato.* Após a inserção de mercados nesta solução novos produtos passam a serem disponibilizados e é refeito o cálculo do custo da solução que tende a diminuir a cada novo item da lista de compras que passa a ser adquirido (disponibilizado na solução parcial), pois o custo dos itens que não podem ser encontrados nos mercados são elevados. Esta provável diminuição do custo ao inserir um mercado representa a sua economia.

Os mercados que possuem menor CUSTO serão os primeiros candidatos a serem inseridos na atual solução. Selecionamos os *p* (no nosso algoritmo usamos *p = 3*) melhores candidatos quem irão compor a lista restrita de candidatos (LCR) (ou menor que *3* se o numero de candidatos total for menor que *3*), de onde será selecionado, de forma aleatória, um mercado a ser inserido na solução corrente. A inserção de um novo mercado se dará até que toda a lista de compras esteja disponível na solução corrente.

### 2.1.2 ADD+DROP Randomizado (ADDR+DROPR)

Este algoritmo baseia-se no uso conjunto dos critérios de inserção e remoção de vértices numa solução parcial. Inicialmente utiliza-se a heurística de inserção randomizada ADDR para o PCV, ou seja, o ADDR é usado até que todos os vértices estejam na solução. Após obtida uma rota do PCV, que contenha todos o vértices, usamos a heurística DROPR para remoção de vértices, cujos passos a serem efetuados são descritos a seguir: A retirada de mercados da solução corrente baseia-se, também, no critério de economia associada a sua remoção, isto é, um vértice será descartado da solução quando sua retirada oferecer maior economia ao custo da solução remanescente (no caso guloso) ou sua retirada oferecer uma das *p* maiores economias onde *p* representa o tamanho da lista restrita de candidatos (no caso DROPR). Os *p* mercados que apresentarem maior economia irão compor a LCR, desde que sua retirada não afete a compra de todos os produtos da lista. Desta LCR, uma será escolhida aleatoriamente para deixar a atual solução. Novos mercados serão excluídos da solução até que mais nenhum possa ser retirado, sem violar a restrição de disponibilidade de todos os produtos nos vértices da solução corrente.

### 2.1.3 Inserção Mais Próxima Randomizada (IMPR)

Esse algoritmo é semelhante ao ADDR, diferenciando-se deste no critério do cálculo da economia, *pois nesse caso o cálculo da economia é baseado no procedimento de inserção do vértice mais próximo da rota parcial,* (e não em relação ao vértice inserido mais recentemente como no ADDR). Neste algoritmo, primeiramente, seleciona-se o vértice que entra na solução e, a seguir, em que posição ele será inserido, enquanto o primeiro (ADDR) avalia a inserção de um novo vértice somente entre o vértice inserido mais recentemente e a origem. Inicialmente a solução é constituída somente da origem. A partir daí, a cada nova inserção, é calculada, a eficiência (número de produtos da lista de compras ainda não adquiridos que passam a ser comprados após a inserção de um mercado) de cada mercado não pertencente à solução. Os *p* mercados de maior eficiência ainda não presentes na solução irão compor a LCR. Um dos mercados da LCR é então aleatoriamente selecionado para ser inserido na solução corrente. É calculada então, a melhor posição para a inserção desse mercado na solução atual a fim de que se tenha o menor custo de translado entre mercados. O critério de parada se dará quando todos os produtos da lista de compras estiverem disponíveis na solução corrente.

### 2.1.4 Inserção Mais Barata Randomizada (IMBR)

Esse algoritmo usa, como critério para inserção na solução corrente, a escolha de um vértice, cuja economia que sua inserção acarreta para a solução seja a maior possível. Logo, o vértice a ser inserido será aquele que represente um menor acréscimo no custo da solução parcial.

*Nesse caso, é levada em consideração somente a média dos preços dos produtos oferecidos em cada vértice,* como critério de seleção para os mercados a serem inseridos na solução parcial.

Para essa versão, o cálculo do custo da solução será o somatório dos custos mais baixos de cada produto da lista de compras, disponível nos mercados pertencentes à solução parcial. Não havendo mercados pertencentes à solução parcial que ofereçam algum dos produtos da lista de compras e estes não puderem ser adquiridos, o seu custo assumirá um valor elevado, que também deverá ser somado no cálculo do custo da solução. Portanto, ao incluir um novo mercado na solução parcial, devemos avaliar qual deles representa uma maior economia à solução parcial, isto é, qual deles proporcionará maior redução ao custo da solução, tornando-a mais barata.

## 2.2 Algoritmos de Busca Local

Os métodos de construção do GRASP, embora tenham a preocupação de sempre gerar soluções de boa qualidade, não representam necessariamente uma solução ótimo local em problemas de otimização. Desta forma é altamente recomendado o uso de um método de busca local para termos a garantia de alcançar ótimos locais de boa qualidade [1, 2, 11, 12, 13, 14]. Apresentamos a seguir, duas estruturas de busca local para os algoritmos GRASP.

### 2.2.1 Busca Local P-trocas

Esta busca local na verdade é um método ADD+DROP (versão tradicional gulosa) usado de forma repetida. A partir da solução inicial, obtém-se uma lista, contendo todos os vértices não pertencentes à solução. Desta lista selecionamos o vértice *k* de modo que sua inclusão na solução corrente acarrete o menor acréscimo ao custo da solução (ou a maior economia).

Inicia-se, então, o processo de remoções sucessivas e seqüenciais, em que serão eliminados da nova solução um a um os vértices, cuja exclusão proporcionarem a maior economia possível à solução remanescente e que, ao mesmo tempo, nenhum produto da lista de compras deixe de ser adquirido. Quando nenhuma melhora puder mais ser obtida, inicia-se novamente o processo de inserir um novo vértice não presente a atual solução. O algoritmo termina quando todos os vértices tiverem ter tido a chance de pertencer à alguma solução intermediaria.

O cálculo da economia associada a inserção de um vértice é baseado no cálculo do custo que este vértice representa para o custo total da solução, no custo da nova distância a ser percorrida e no número de produtos da lista de compras, que deixariam de ser adquiridos após a retirada desse vértice.
**Critério de inserção:** O novo mercado inserido na solução é aquele que possuir a *menor* média de preços para os produtos da lista de compras a serem adquiridos e que ainda não pertença à solução.
**Critério de retirada:** Retiram-se mercados da solução parcial até que nenhum mercado possa ser retirado sem que algum produto da lista de compras deixe de ser adquirido, para tal, o mercado a ser retirado é aquele que apresentar maior economia à solução.

### 2.2.2 Busca Local VNS

Essa busca se baseia na idéia da Metaheurística *Variable Neighborhood Search* (VNS) [5, 12, 14], em que uma sistemática troca de vizinhança é realizada durante a busca local. No entanto modificações são feitas em relação ao VNS tradicional para não onerar os tempos computacionais desta busca local dentro da estrutura de um GRASP. Melhor exemplificando, temos: seja I o conjunto de mercados pertencentes a solução corrente e J o conjunto de mercados fora da solução corrente. Cada vizinhança Vi (.) é da forma:

$V_1(.)$ = *Permutação entre cada par* $(i, j)$, $\forall\, i \in I$ *e* $\forall\, j \in J$.
$V_2(.)$ = *Permutação entre dois pares distintos* $(i_1, j_1)$ *e* $(i_2, j_2)$, $\forall\, (i_1 \text{ e } i_2) \in I$ *e* $\forall\, (j_1 \text{ e } j_2) \in J$.
$V_3(.)$ = *Permutação entre três pares distintos* $(i_1, j_1)$, $(i_2, j_2)$ *e* $(i_3, j_3)$, $\forall\, (i_1, i_2 \text{ e } i_3) \in I$ *e* $\forall\, (j_1, j_2 \text{ e } j_3) \in J$.

Uma solução é investigada em uma vizinhança até que não se consiga mais obter melhoras, então, passa-se a explorar a partir da atual semente a próxima estrutura de vizinhança. Ao final da busca, a melhor solução, obtida entre todas as trocas realizadas, é considerada como a nova solução. Desta forma, não retornamos a primeira estrutura sempre que uma melhora for obtida numa vizinhança como é feito no VNS tradicional.

## 2.3 Combinações dos Algoritmos

A partir dos algoritmos de construção e busca local propostos para o PCS, foram desenvolvidas inicialmente 8 versões da metaheurística GRASP, descritas na Tabela 1.

| Algoritmo | Construção | Busca Local |
|---|---|---|
| G1 | ADDR | VNS |
| G2 | ADDR+DROPR | VNS |
| G3 | IMPR | VNS |
| G4 | IMBR | VNS |
| G5 | ADDR | P_trocas |
| G6 | ADDR+DROPR | P_trocas |
| G7 | IMPR | P_trocas |
| G8 | IMBR | P_trocas |

Tabela 1: versões da heurística GRASP propostos

Além das 8 versões iniciais para o GRASP, foram implementadas variações destas, utilizando a técnica de filtro na etapa de construção e, para as melhores versões resultantes de uma bateria de testes inicial, é proposto a técnica de reconexão de caminhos na etapa de busca local.

### 2.4 GRASP com Filtro

Nas versões G1 a G8, constrói-se a cada iteração GRASP, uma única solução e nesta é aplicada a busca local. O que difere o GRASP com filtro destes modelos, é que nas versões com filtro, o algoritmo de construção é executado *x* vezes, (no nosso trabalho usamos *x = 50*), construindo-se, assim, *x* soluções iniciais diversificadas. Dessas *x* soluções iniciais, selecionamos somente a melhor delas para a etapa da busca local. Logo, com uma solução inicial de melhor qualidade, a solução final obtida após a busca local também tende a ser melhor. Portanto incluindo o filtro na construção do GRASP, ao final, teremos a partir da tabela 1, não 8, mas 16 versões GRASP acrescentando para cada GJ, sua versão com filtro GJF.

### 2.5 GRASP com Reconexão de Caminhos

O uso da estratégia GRASP com reconexão de caminhos (*path relinking*) tem sido usado com muito sucesso na solução de diferentes problemas da área de otimização combinatória. O conceito de reconexão de caminhos (RC) foi originalmente proposto por Glover no método Busca Tabu (veja em [3]). Posteriormente tem sido usado para GRASP, Algoritmos Evolutivos e outras metaheuristicas [1, 2, 11, 13]. A idéia central da RC, é a partir de duas soluções de boa qualidade, analisar todas as soluções intermediárias entre elas. A justificativa para isso, é a de que entre duas soluções extremas de qualidade possa existir uma terceira melhor que ambas. Para uso deste módulo, normalmente exige-se que se armazene, não apenas a melhor solução gerada até o momento pela heurística, mas um conjunto P das h melhores soluções distintas (onde h é um parâmetro de entrada). Este conjunto P é denominado *conjunto elite* e suas componentes *soluções elite*. No nosso trabalho a RC é usada da seguinte forma para a solução do PCS:

Inicia-se a execução do algoritmo com o GRASP, em que novas soluções são encontradas a cada iteração através das fases de construção e busca local. Durante as *r* primeiras iterações seleciona-se para o conjunto elite *P*, as *h* melhores soluções obtidas durante essas iterações. A cada nova solução obtida, compara-se àquelas armazenadas no conjunto *P*, e, sempre que uma nova solução obtida for melhor que qualquer do conjunto elite, *P* é atualizado. A partir da (*r+1*) - ésima iteração, a reconexão de caminhos é ativada. A solução corrente obtida nessa iteração será usada como *solução base* e as já armazenadas no conjunto elite *P* serão usadas como *soluções alvo*. No nosso trabalho usamos *r =50;e h = 3*. Inicia-se, então, o processo de conexão entre a solução base e a solução alvo, através da permutação entre seus mercados. Esse procedimento inicia determinando todos os mercados que não sejam comuns às duas soluções, isto é, aqueles que pertencem à solução base e não esteja na solução alvo, aqui chamados de $m_{fa}$ e, da mesma forma, os que pertençam a solução alvo e que não estejam na solução base, chamados aqui de $m_{fb}$. Faz-se, então, uma permutação entre os pares ($m_{fb}$, $m_{fa}$), gerando um movimento da base em direção ao alvo. Esse movimento consiste da retirada, um a um, dos mercados $m_{fb}$ selecionados, e da substituição deste por cada um dos mercados $m_{fa}$. As permutações são executadas até que a solução base fique idêntica à solução alvo.

A reconexão de caminhos (RC) proposta neste trabalho explora, não só a trajetória da solução base em direção à solução alvo (*forward*), mas também a trajetória inversa, da solução alvo em direção à solução base *(backward)* que se dará de forma análoga. O procedimento de reconexão é executado da mesma solução base para cada uma das soluções alvo armazenadas no conjunto elite *P*. A cada permutação, são avaliadas a validade e a qualidade da nova solução obtida, e, sempre que esta seja melhor que qualquer uma das constantes no conjunto elite *P*, ele é atualizado, após o término deste procedimento. Após a primeira ativação da reconexão de caminhos, uma nova ativação se dará sempre que (*x* %) do conjunto elite tenha sido atualizado (onde *x* é um dado de entrada) ou ainda, após *r* iterações consecutivas sem ativação da RC. Neste trabalho aplicamos a estratégia de reconexão de caminhos somente nas duas melhores versões, dentre as *16* propostas para o do GRASP, elevando-se assim o número final de versões GRASP para *18*.

## 3. Implementação e Resultados Computacionais

Para verificar o desempenho dos algoritmos propostos e possibilitar uma análise comparativa no seu desempenho, diferentes tipos de testes computacionais são realizados. Para a realização desses testes, quando existentes, devem ser utilizadas bibliotecas públicas de problemas testes, cuja melhor solução é conhecida, e, através da qual, pode-se comparar e avaliar os resultados obtidos pelos métodos propostos. No caso do problema PCS explorado neste trabalho, apesar de suas inúmeras aplicações, não existe muitos métodos na literatura e além disso não foi encontrado nenhuma biblioteca pública de problemas testes. Esta limitação se verifica mesmo para os trabalhos desenvolvidos anteriormente por pesquisadores daqui do Brasil onde infelizmente os autores não guardaram as instancias utilizadas e nem os códigos desenvolvidos. Assim, a fim de se poder testar e avaliar a qualidade dos algoritmos propostos, novos problemas-teste foram gerados artificialmente da seguinte forma: Cada vértice sempre pertence ao espaço $R^2$. As coordenadas de cada vértice (mercado) foram geradas aleatoriamente num intervalo pré-definido, e as distâncias entre os vertices foram calculadas através da métrica euclidiana para cálculo das distâncias. Os custos dos diversos produtos, em cada mercado, foram gerados aleatoriamente também dentro de um intervalo pré-definido. Foi analisado um total de *85* instâncias . As instâncias, utilizadas para a realização dos testes, foram geradas com as características a seguir: o número de vértices (mercados) variou de 15 a 200; o número de produtos disponíveis em cada mercado variou entre 20 e 1000 unidades; e o intervalo dos custos de compra e translado, que serão descritos a seguir. A nomenclatura utilizada para representar cada instância possui as seguintes características: **I_Mi_Pi_Ini**, onde $M_i$ representa o nº de vértices da instância; $P_i$ variedade (número) de produtos que estão disponíveis nos mercados; e $In_i$ que é o intervalo adotado, podendo assumir os valores 1, 2 e 3. Quando $In_i$ é igual a 1, indica que os custos dos produtos variaram no intervalo entre [10, 100], quando $In_i$ assume o valor 2 indica que os custos de compras dos produtos variaram no intervalo entre [30, 150] e, quando $In_i$ é 3, indica que intervalo variou entre [50, 200]. O intervalo utilizado para gerar as coordenadas de cada mercado, necessárias para o cálculo das distâncias, variaram entre [0, 50], logo o custo de translado também variou entre [0, 50]. Para cada uma das instâncias descritas na tabela 2, foram testadas inicialmente as 16 versões (8 versões GRASP sem e com filtro) mostradas anteriormente e cada instância foi executado por três vezes por cada algoritmo. O critério de parada de cada execução foi após 500 iterações GRASP.

### 3.1 Testes realizados

Nesta seção, serão mostrados os testes computacionais realizados e o desempenho obtida pelos mesmos. Inicialmente mostraremos, os resultados obtidos nos testes realizados que combinaram as primeiras 16 versões do GRASP (sem RC) com as 85 instâncias consideradas. Nas simulações efetuadas nesta fase, foram analisadas para cada instância considerada, o melhor resultado e o tempo de execução médio obtido em 3 execuções por cada algoritmo. Devido a limitação no tamanho deste artigo, não ilustramos todos os resultados de cada algoritmo em cada uma das 85 instâncias consideradas. De forma simplificada a tabela 2 mostra o número de vezes em que cada algoritmo atingiu a melhor solução (*best)* no conjunto das instâncias consideradas. Para a ordenação, priorizamos

o desempenho na seguinte ordem: *melhor média obtida em cada instância; melhor solução obtida em cada instância e como DF (desempenho final, a soma destes dois números como descrito na tabela 3).*

| Versão | Nº de vitórias | | DF |
|---|---|---|---|
| | Melhor Média | Melhor solução | |
| **G2F** | 27 | 24 | 51 |
| **G2** | 17 | 23 | 40 |
| **G3F** | 14 | 17 | 31 |
| **G3** | 11 | 12 | 23 |
| **G1F** | 11 | 8 | 19 |
| **G4F** | 5 | 4 | 9 |
| **G4** | 4 | 4 | 8 |
| **G1** | 1 | 3 | 4 |
| **G6F** | 2 | 2 | 4 |
| **G6** | 2 | 2 | 4 |
| **G5F** | 0 | 1 | 1 |
| **G8F** | 0 | 1 | 1 |
| **G5** | 0 | 0 | 0 |
| **G7F** | 0 | 0 | 0 |
| **G7** | 0 | 0 | 0 |
| **G8** | 0 | 0 | 0 |

Tabela 2: Classificação dos algoritmos segundo o número de vitórias obtidas nas 85 instâncias.

Os resultados da tabela 2 nos mostram o seguinte. Ao menos nos testes empíricos aqui realizados; e em termos absolutos, as versões G2F e G2 foram as mais eficientes seguidas de perto por G3F, G3 e G1F. Através de todos os resultados mostrados, podemos concluir que as versões que utilizam a metaheurística VNS em sua fase de busca local, apresentam resultados superiores aos obtidos nas demais versões. Ainda em termos absolutos, na outra ponta estão os de pior desempenho (nenhuma vitória ou DF = 0) que são: G7, G8, G5, G7F. Um aspecto importante, é que em todos os casos de algoritmos que obtiveram alguma vitória, a versão com filtro sempre foi superior a sua versão sem filtro. Isso mostra a importância de usarmos este procedimento para tentar soluções de melhor qualidade. Em termos comparativos globais o algoritmo G2 foi a melhor (considerando G2 e G2F). Isto é, a melhor combinação foi o construtivo ADDR+DROPR com busca local VNS. Outro aspecto importante, das melhores versões (G2, G3, G1) todas utilizam a busca VNS, mostrando como o VNS se comporta muito bem na estrutura GRASP. Em termos do método de construção a melhor foi o ADDR+DROPR, seguido pelo IMPR. Mas aqui não podemos afirmar que isoladamente estes construtivos são os mais eficientes, e sim o que podemos concluir é que a *combinação destes com o VNS* trouxe sim os melhores resultados. Do outro lado, os piores resultados sempre usam a busca P trocas. Obviamente a busca VNS exige um tempo computacional maior que o P trocas como comentado mais adiante, mas em termos de qualidade da solução gerada, a busca VNS mostrou sem nenhuma dúvida sua nítida superioridade. Depois dessa bateria de testes, selecionamos dois algoritmos para serem utilizados na estratégia de reconexão de caminhos (RC). Apesar de havermos concluído que as melhores versões, em termos de resultados obtidos, foram respectivamente o G2F e o G2, devido à similaridade apresentada entre estas duas versões, decidimos selecionar, além do G2F, a versão G3F para o lugar do G2, pois concluímos que versões mais diversificadas poderiam apresentar uma melhor diversificação nos resultados.

| Sem Reconexão de Caminhos | | | | Com Reconexão de Caminhos | | | |
|---|---|---|---|---|---|---|---|
| G3F | | G2F | | G3F+RC | | G2F+RC | |
| **Melhor** | **Média** | **Melhor** | **Média** | **Melhor** | **Média** | **Melhor** | **Média** |
| 0 | 0 | 0 | 0 | **67** | **54** | 22 | 31 |

Tabela 3: Resumo do desempenho (número de vezes em que se chegou a melhor solução (melhor média conhecida) para cada uma das 85 instâncias.

Na tabela 3, são mostrados os resultados obtidos com as melhores variações do GRASP e estes com a aplicação de reconexão de caminhos. Analisando a tabela, podemos concluir que as versões que utilizaram a estratégia de reconexão de caminhos melhoraram os resultados obtidos *em todas as 85 instâncias analisadas*. Concluímos, então, que essa estratégia é extremamente promissora, podendo apresentar melhoras significativas na qualidade das soluções alcançadas. Comparando as duas versões analisadas com RC observamos que o melhor desempenho foi obtido pelo G3F+RC cuja superioridade sobre o G2F+RC é notória. Além disso, o G3F+RC se mostrou *muito mais econômico* em relação aos tempos gastos. Os resultados dos tempos absolutos e médios não são aqui ilustrados devido novamente a limitação do tamanho deste artigo, mas para se ter uma idéia da diferença entre os tempos, nas instâncias de tamanho 1000 vértices, o tempo médio do G3F+RC foi de no máximo 10 segundos e o tempo médio do G2F+RC girou em torno dos 78 segundos usando como critério de parada o número máximo de iterações igual a 500. Podemos, então, afirmar que tanto quando avaliamos os tempos computacionais utilizados, quanto a qualidade do algoritmo, o G3F se adaptou melhor a estratégia de RC, apresentando os melhores resultados finais. É interessante observar que sem a RC as melhores versões até então tinham sido o G2F e G2. Ainda em relação aos tempos computacionais exigidos, comparando versões sem RC e com RC, os resultados mostram que obviamente com RC, existe um aumento de tempo, mas estes são pequenos (no máximo de 15% dos tempos globais). Esta questão não é mostrada explicitamente em tabelas, porque entendemos que uma melhor interpretação pode ser feita na analise probabilística sugerida a seguir.

### 3. 2 Análise Probabilística Empírica

Um gargalo observado em alguns algoritmos metaheurísticas da literatura é a sua instabilidade ou a falta de regularidade em seu desempenho. Em outras palavras, é comum ocorrer que uma determinada heurística apresente um bom desempenho para uma classe de problemas ou mais especificamente para algumas instâncias, mas para outras o seu desempenho é ruim.

Isto justifica a proposta de uma nova bateria de testes, agora com o objetivo específico de analisar a robustez dos algoritmos GRASP aqui apresentados. O critério de parada dos algoritmos é modificado para que a execução prossiga até que uma solução de valor igual ou melhor que um valor alvo estabelecido tenha sido encontrado. Em cada execução $i$, o tempo $t_i$ para atingir o valor alvo é armazenado em ordem crescente. Uma probabilidade empírica $p_i = (i - 0{,}5)/100$ é associada a cada tempo $t_i$. Desta forma, são plotados os pontos $z_i = (t_i, p_i)$, estabelecendo uma distribuição empírica de tempo para que um algoritmo alcance um determinado valor alvo [1].

Neste trabalho, os valores alvo serão estabelecidos, procurando alcançar três patamares distintos. No primeiro, que chamaremos de *alvos fáceis*, o valor do alvo é calculado através da média dos piores valores obtidos pelas versões analisadas. O *alvo médio* que será obtido através da média aritmética de todos os resultados obtidos pelas versões avaliadas. E um *alvo difícil*, será calculado pela média dos três melhores resultados obtidos considerando todas as versões analisadas.

Na avaliação das 16 primeiras versões propostas, aquelas que não utilizaram RC, devido à diversidade da qualidade dos resultados obtidos por cada uma delas, nem todas as versões foram testadas para os três tipos de alvo estabelecidos, pois nem todas as versões teriam a capacidade de alcançar alvos mais difíceis dentro de tempos computacionalmente viáveis. Para os alvos médios, foram testadas somente as versões G2, G2F, G3, G3F e G1F, e a escolha dessas versões baseou-se no desempenho verificado nas análises prévias. De forma análoga, para alvos difíceis, foram testadas somente as versões G2, G2F e G3F. No caso das versões que utilizaram a técnica de RC, sua análise probabilística foi feita comparando seus resultados com os alcançados pelas mesmas versões sem o uso deste módulo. Interessa-nos, principalmente nesse caso, para fins comparativos, computar o tempo que cada versão levou para atingir o alvo estabelecido. A fim de analisar esses tempos e poder efetuar conclusões significativas, cada estância foi executada 100 vezes, para cada uma das combinações entre as versões explicitadas anteriormente, dos algoritmos e os alvos a serem alcançados. Devido à semelhança nos resultados obtidos, e a limitação no tamanho deste artigo, selecionamos apenas alguns casos para ilustrar esta análise.


Figura 1: análise probabilística empírica de algumas versões propostas usando *alvo fácil*

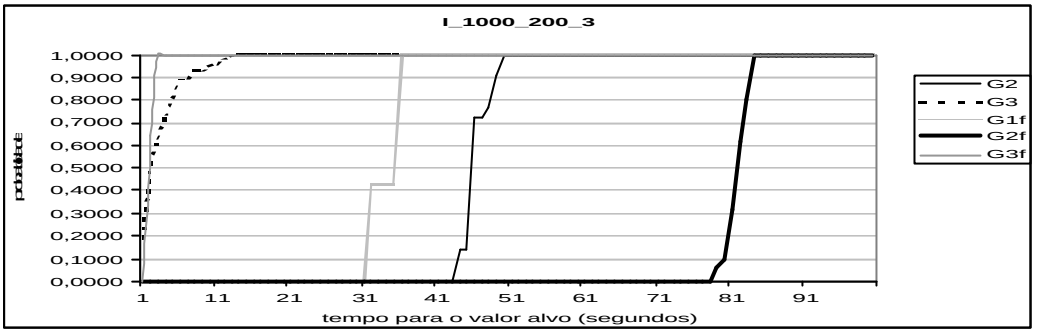


Figura 2: análise probabilística empírica de algumas versões propostas usando *alvo médio*

Na figura 1, temos uma comparação da convergência para alvos fáceis entre as 8 melhores versões encontradas na tabela 2. Na figura 2, ilustra a convergência dos 5 melhores GRASP para alvo médio. Nestas figuras cada curva de cada figura representa um algoritmo. O eixo horizontal representa o tempo em segundos gastos por cada algoritmo para atingir determinada taxa de convergência para o valor alvo. O eixo vertical mostra a taxa de convergência obtida em cada tempo por cada algoritmo. Assim, concluímos que "*quanto mais a esquerda estiver a curva*" melhor (mais rápida) será a convergência do algoritmo. Por exemplo, na figura 1, o melhor desempenho foi do G3F (curva mais a esquerda), seguido pelo G2, e G3 sendo que o pior resultado (convergência mais lenta) foi do G1f (curva mais a direita). Os testes foram efetuados para todas as instâncias, mas somente este caso é aqui ilustrado devido à similaridade nos resultados. Na figura 2, num alvo médio, observamos que G3 apresenta o melhor desempenho seguido do G2. O G3 necessita de somente pouco mais de 11 segundos para atingir uma taxa de convergência empírica de 100%. Ou seja, neste tempo todas as 100 execuções de G3 atingiram o alvo estabelecido. Enquanto isso, versões mais lentas como o G2F neste caso necessitam de cerca de 81 segundos para atingir esta taxa.

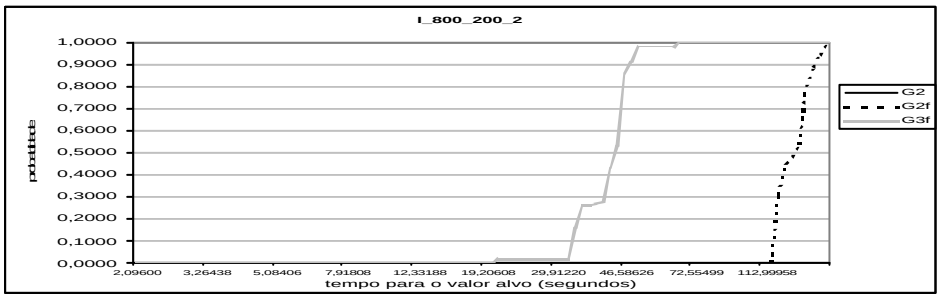


Figura 3: análise probabilística empírica da instância I_800_200_2 das 3 melhores versões propostas usando alvo difícil (*neste caso a curva de G2 não aparece devido a sua não convergência num tempo máximo estabelecido)*

A figura 3 ilustra a comparação entre as 3 melhores versões, capazes de atingir o alvo mais difícil. Nestes testes, os demais algoritmos não foram analisados (plotados nos gráficos) devido à dificuldade destes em convergir para estes alvos difíceis num tempo computacional máximo permitido. Através da análise observamos que a versão G3F, alcança a solução alvo mais rapidamente que as demais. Ao contrário dessa, a versão G2F apresenta em todos os gráficos a convergência mais lenta entre os aqui analisados. Nesta figura 3, o G2 não conseguiu atingir o alvo na grande maioria das 100 execuções, e desta forma foi também retirada desta análise.

Os valores comparativos entre a qualidade dos resultados obtidos mostrados anteriormente nas tabelas 2, 3 e a análise dos gráficos de convergência empírica (considerando todas as instâncias analisadas e não somente as ilustradas neste trabalho) nos permitem afirmar que a versão G3 ou G3F, embora tenha obtido apenas o terceiro lugar na tabela 2 (G3F) nesta análise converge sempre mais rápido para valores alvo fáceis, médios e difíceis quando comparado com a versão G2F, que obteve o 1º lugar (tabela 2). Esta afirmação nos permite concluir que dependendo do critério de parado usado, a versão G3F pode apresentar resultados de qualidade ligeiramente inferiores que a melhor versão, mas que quando usados como critérios de parada um valor alvo, G3F ou G3 sempre alcança os melhores resultados mostrando com isso uma melhor robustez que outras versões.

Adicionalmente foi avaliado também como se comportam as versões que utilizam a estratégia de reconexão de caminhos (Figura 4), comparando-as com as mesmas versões sem a reconexão, verificando, que as versões ***com RC*** convergem mais rapidamente para os valores alvo, independente da instância considerada. Novamente devido a similaridade nos resultados e devido a limitação no tamanho deste artigo, somente ilustramos o caso de uma instancia usando um alvo difícil.


Figura 4: análise probabilística empírica dos 2 melhores versões propostas sem e com reconexão de caminhos usando alvos difíceis

Os resultados da análise probabilística com alvos difíceis considerando todas as estâncias, nos mostram outro potencial das versões que incluem o módulo RC. Embora estes tenham nos testes anteriores apresentado tempo computacional maior que outras versões, se usarmos como critério de parada, o alcance de um valor alvo sub-ótimo, as versões com RC, também se mostram os mais rápidos e mais robustos reforçando novamente a eficiência destes.

Nesta análise probabilística, observamos que com este novo critério de parada nem sempre a versão considerada a *mais pesada* nos testes onde o critério de parada foi o numero de iterações; tem a *convergência mais lenta.* Isso fica claro nos resultados das figuras 1 a 4, onde versões com filtro (ou com reconexão de caminhos) quase sempre são mais rápidos que seu similar sem o filtro (ou sem a reconexão). Isso mostra o cuidado que devemos ter na definição do critério de parada nestes métodos pois dependendo do critério usado, o seu desempenho pode ser bastante prejudicado.

## 4. Conclusões

O objetivo deste trabalho foi a de analisar o impacto de heurísticas de construção e busca local na metaheuristica GRASP utilizado para a solução de um Problema de Coleta Seletiva. Propomos então o uso do GRASP, com variações nos procedimentos de construção e busca local. Na fase de

construção, utilizamos adaptações de técnicas já bastante conhecidas como ADD, DROP, Inserção mais próxima e Inserção mais barata, adaptando-as para o GRASP aplicado ao PCS, e ainda o uso da metaheurísticas tipo VNS e da técnica P_trocas para efetuar busca local.

Utilizamos também um procedimento construções sucessivas e filtragem de soluções iniciais, o que chamamos de Filtro. Esse procedimento foi aplicado a todas as combinações propostas e, para todas elas, melhorou os resultados obtidos. Para chegar a essa conclusão, experimentos computacionais extensivos, aplicados a instâncias pequenas, médias e grandes, foram feitos para as variações do GRASP. Com isso, podemos então, identificar as melhores versões dentre todas as propostas.

Para essas melhores versões, foi também introduzida a técnica de reconexão de caminhos, que permitiu melhorar ainda mais o desempenho do algoritmo GRASP, fazendo com que soluções de melhor qualidade pudessem ser encontradas mais rapidamente.

Ainda com a reconexão de caminhos, testes extensivos foram executados, e constatamos que ela proporcionou uma melhoria significativa na qualidade das soluções encontradas em todas as instancias avaliadas, além de se apresentar como as versões mais robustas em relação a convergência para valores alvos sub-ótimos. Através dos resultados computacionais obtidos, podemos também concluir que resultados heurísticos mais promissores foram obtidos por versões híbridas, que conjugaram conceitos de GRASP, VNS , Filtro e Reconexão de Caminhos num único algoritmo. Esses resultados nos mostram o enorme potencial dessas técnicas na solução de problemas de elevada complexidade computacional, como é o caso do PCS.

## 5. Referências Bibliográficas


[1] AIEX, R., BINATO, S., e RESENDE, M. *Parallel GRASP with path-relinking for job shop scheduling*. Parallel Computing 29, pp. 393-430, 2003.

[2] FESTA, P., RESENDE, M. *GRASP: An annotated bibliography*. In Ribeiro, C. Hansen, P., editors. Essays and Surveys in Metaheuristics, pp. 325-367. Kluwer Academic Publishers, 2002.

[3] GLOVER, F.; LAGUNA, M. *Fundamentals of Scatter Search and Path Relinking*. Control and Cybernetics. Vol. 29, no. 3, pp. 653-684, 2000.

[4] GOLDEN, B. L., LEVY, L. e Dahl, R. *Two generalizations of the TPP*, OMEGA, pp. 439-445, 1981.

[5] HANSEN, P. e MLADENOVIC, N. *Variable Neighborhood Search for the -pmedian*. Location Science 5, pp. 207-226, 1998.

[6] ONG, H. L., *Approximate Algorithms for the Traveling Purchaser Problem*, Operations Research Letters, Vol. 1, no. 5, pp. 201-205, 1982.

[7] PEARN, W. L. *The Traveling Purchaser Problem*, Department of Industrial Engeneering and Management, National Chiao Tung University,1991.

[8] PEARN, W. L., Chien, R. C. *Improved Solutions for the Traveling Purchaser Problem, Computers Operation Research*, Vol. 25, no. 11, pp. 879-885, 1998.

[9] RAMESH, T. *Traveling Purchaser Problem.* OPSEARCH, Vol. 18, no. 2, pp. 78-91, 1981.

[10] RAVI, R, e SALMAN, F. S. *Approximation Algorithms for the Traveling Purchaser Problem and its Variants in Network Design*, GSIA, Carnegie Mellon University, Pittsburgh.

[11] RESENDE, M.G.C. e FEO, T.A. *Greedy Randomized Adaptative Search Procedures*. Journal of Global Optimization 6, pp. 109-133, 1995.

[12] SILVA, M. B., OCHI, L. S. & DRUMMOND, L. M. A. *Metaheuristics based on GRASP and VNS for solving The Traveling Purchaser Problem;* Proc. of the IV Metaheuristic International Conference (MIC'2001), pp: 489-494, Porto, Portugal, 2001; Editors**:** Ana Vianna and J. P. Souza.

[13] SILVA C., OCHI. S., & MARTINS, S. L. *Experimental Comparison of Greedy Randomized Adaptive Search Procedures for the Maximum Diversity Problem. Lecture Notes on Computer Science* 3059, 498-512, 2004, SPRINGER.

[14] VIANNA, L. *Metaheurísticas Seqüenciais e Paralelas para o TPP*. Dissertação de mestrado. UFF, Niterói, Departamento de Ciência da Computação, 2003.

[15] VOSS S. *ADD and DROP-procedures for the traveling purchaser problem.* Methods of operations research, 53, pp. 317-318, 1996.