APLICAÇÃO DO ALGORITMO CLUSTERING SEARCH AOS TRAVELING SALESMAN PROBLEMS WITH PROFITS

Antonio Augusto Chaves
Instituto Nacional de Pesquisas Espaciais (INPE)
Av. dos Astronautas, 1758 São José dos Campos – SP – Brasil
chaves@lac.inpe.br

Luiz Antonio Nogueira Lorena
Instituto Nacional de Pesquisas Espaciais (INPE)
Av. dos Astronautas, 1758 São José dos Campos – SP – Brasil
lorena@lac.inpe.br

RESUMO

Traveling Salesman Problems with Profits (TSPP) são generalizações do Problema do Caixeiro Viajante (PCV), onde não é necessário visitar todos os vértices. Para cada vértice é associado um prêmio. O objetivo é otimizar simultaneamente o prêmio coletado e os custos de deslocamento. Estes dois critérios de otimização podem aparecer juntos na função objetivo ou como uma restrição. Este trabalho propõe uma nova abordagem para resolver alguns dos *TSPP*, utilizando uma metaheurística híbrida denominada *Clustering Search* (CS). A validação das soluções obtidas será através da comparação com os resultados encontrados pelo CPLEX.

PALAVRAS CHAVE. Problema do Caixeiro Viajante. Metaheurística. Formulação Matemática. Otimização Combinatória.

ABSTRACT

Traveling Salesman Problems with Profits (TSPP) are generalizations of the Traveling Salesman Problem (TSP) where it is not necessary to visit all vertices. The goal is to optimize the collected profit and the travel costs simultaneously. These two optimization criteria can appear in the objective function or as a constraint. This paper proposes a new approach called Clustering Search (CS) to solve some TSPP using hybrid metaheuristics. The validation of the obtained solutions will be through the comparison with the results found by the CPLEX.

KEYWORDS. Traveling Salesman Problems. Metaheuristics. Clustering Search. Combinatorial Optimization.

1. Introdução

O Problema do Caixeiro Viajante (PCV) referido na literatura como *Traveling Salesman Problem* (TSP) (Reinelt, 1994) é um dos mais tradicionais e conhecidos problemas de otimização combinatória. O PCV consiste em otimizar a seqüência de visitas a clientes a partir de um depósito central, sendo que, uma característica deste problema é que todos clientes precisam ser visitados, e conseqüentemente, nenhum valor é associado ao serviço de atendimento ao cliente. Porém, algumas generalizações deste problema propõem selecionar clientes dependendo de um valor de prêmio (benefício) que é ganho quando a visita acontecer. Esta característica dá origem a uma classe de problemas que na literatura foi nomeada *Traveling Salesman Problems with Profits* ou Problemas do Caixeiro Viajante com Lucros (TSPP) (Feillet *et al*., 2006).

TSPP podem ser visto como um problema do caixeiro viajante bi-objetivo com dois objetivos opostos, um que pressiona o caixeiro a viajar (ou seja, coletar prêmios) e outro que estimula o caixeiro a minimizar os custos de viagem (permitindo a ele não visitar clientes). Na prática, a maioria das pesquisas existente sobre estes problemas os trata como sendo versões de único objetivo. Assim, ou os dois objetivos são calculados e combinados linearmente, ou um dos objetivos é transformado em restrição com um valor limite especificado.

Os TSPP são de fácil adaptação a situações da vida real, tendo várias aplicações práticas e de relevância econômica. Em linhas gerais, pode ser associado a um universo de clientes em potencial, a um conjunto de cidades ou a pontos turísticos a serem visitados, onde se deseja determinar quem deve ser visitado e qual a seqüência das visitas.

A dificuldade de solução dos TSPP está no número elevado de soluções existentes, uma vez que, estes problemas pertencem à classe de problemas NP-difíceis. Isto é intuitivamente claro, desde que, uma solução do PCV pode ser declarada como uma solução de um TSPP onde precisa-se coletar todos os prêmios dos vértices.

Este trabalho aborda alguns dos problemas da classe TSPP, tais como, *Prize Collecting Traveling Salesman Problem* (PCTSP) (Balas, 1989), *Profitable Tour Problem* (PTP) (Dell'Amico *et al.*, 1995) e *Quota Traveling Salesman Problem* (QTSP) (Awerbuch *et al*. 1998). Para a resolução destes problemas, propõe-se uma abordagem heurística que faz uso de conceitos de uma metaheurística híbrida proposta recentemente por Oliveira e Lorena (2007), chamada *Clustering Search* (CS) que consiste em detectar dinamicamente regiões promissoras de busca baseando-se na freqüência em que são amostradas nestas regiões as soluções geradas por uma metaheurística. Estas regiões promissoras devem ser exploradas tão logo sejam descobertas, através de métodos de busca local.

O *software* CPLEX (Ilog, 2001) é utilizado para resolver as formulações matemáticas do PCTSP, PTP e QTSP, objetivando validar os resultados computacionais obtidos pela abordagem CS. Sendo o CPLEX capaz de resolver apenas problemas-teste de pequeno porte.

O restante do trabalho está organizado da seguinte forma. A seção 2 faz uma descrição dos *TSPP* e apresenta algumas formulações matemáticas para estes. A seção 3 descreve o CS, e a seção 4 apresenta, em detalhes, o CS aplicado a alguns problemas da classe *TSPP*. A seção 5 apresenta os resultados computacionais e na seção 6 são descritas algumas conclusões a respeito desse trabalho.

2. TSPs with Profits

TSPs with Profits (TSPP), de uma forma geral, podem ser representados em um grafo completo G=(V,A), onde V = $\{v_0, v_1,...,v_n\}$ é um conjunto de *n* vértices e A é um conjunto de arestas (grafo não direcionado). Para cada vértice $v_i \in$ V existe associado um prêmio p_{v_i}, e cada aresta $(v_i, v_j) \in$ A possui um custo de deslocamento $c_{v_i v_j}$. Supondo que o vértice v_0, sem perda de generalidade, seja o depósito ou a cidade de origem do caixeiro, este vértice deve ter prêmio nulo ($p_{v_0} = 0$). TSPP consiste em determinar um circuito elementar que contenha o vértice v_0, levando em consideração o prêmio coletado e os custos de deslocamento.

Os diferentes problemas que formam o TSPP surgem das diferentes maneiras que

existem para tratar os dois objetivos:

1. Os dois objetivos são tratados separadamente, gerando um problema multi-objetivo, onde os objetivos são, minimizar os custos de deslocamento e maximizar os prêmios coletados. Este problema é conhecido como *Multiobjetive Vending Problem* (MVP) (Keller e Goodchild, 1988);
2. Ambos os objetivos são combinados numa função objetivo, visando encontrar uma rota que minimize os custos de deslocamento menos os prêmios coletados. Dell'Amico *et al.* (1995) tratam esta versão como sendo o *Profitable Tour Problem* (PTP), sendo que, ao invés de coletar um prêmio por visitar uma cidade, o caixeiro paga uma penalidade caso deixe de visitar alguma cidade.
3. O objetivo do custo de deslocamento é definido como uma restrição, e o objetivo é encontrar uma rota que maximize o prêmio coletado tal que o custo de deslocamento não exceda um valor máximo. Este problema é chamado de *Orienteering Problem* (OP) (Tsiligirides, 1984). Existem na literatura alguns trabalhos que consideram esse problema como sendo um circuito, classificando-o como *Selective TSP* (STSP) (Laporte e Martello,1990) ou Maximum Collection Problem (Kataoka e Morito, 1988);
4. O objetivo do prêmio é definido como uma restrição, e o objetivo é encontrar uma rota que minimize os custos de deslocamento e que o prêmio coletado não seja menor que um valor pré-definido. Este problema é definido como *Prize-Collecting TSP* (PCTSP) (Balas, 1989), onde também é inserido o conceito de penalidades, ou *Quota TSP* (QTSP) (Awerbuch *et al.*, 1998).

Neste trabalho foi abordado três destes problemas, sendo eles: PCTSP, PTP e QTSP. Estes problemas foram resolvidos através da metaheurística CS, sendo que, para todos os problemas a implementação do CS é exatamente a mesma e os parâmetros utilizados também, a única diferença de implementação está no cálculo da função objetivo de cada problema.

3. Revisão Bibliográfica

O PCTSP foi proposto por Balas (1989), que apresentou algumas propriedades estruturais do problema e duas formulações matemáticas para este, sendo a segunda uma simplificação da primeira. O autor apresenta também um algoritmo para o problema, que combina a técnica de planos de corte com uma relaxação de programação linear a ser resolvida através do método *simplex*.

Dell'Amico *et al.* (1995) apresentam uma nova abordagem para obter *lower bounds* para o PTP e PCTSP. Utilizando a formulação apresentada em (Balas, 1989) eles propuseram uma relaxação Lagrangeana, relaxando a restrição de prêmio mínimo, e usam o método subgradientes para resolver o problema dual.

Awerbuch *et al.* (1998) propuseram um algoritmo aproximativo para o QTSP com performance $O(\log^2 n)$, para isso utilizaram um algoritmo que agrupava os pontos do problema dentro de componentes, assim como o algoritmo de Kruskal, e procurava unir (usando o caminho mais curto) os dois componentes que apresentassem a menor razão entre a distância entre eles e o número mínimo de pontos dentro dos dois componentes.

Chaves *et al.* (2004) exploraram duas abordagens para o PCTSP, a primeira é uma abordagem exata capaz de resolver problemas-teste de pequenas dimensões, e a segunda é uma abordagem heurística, combinando as metaheurísticas GRASP e VNS/VND. Chaves e Lorena (2005) propuseram novas heurísticas baseadas no CS para resolver o PCTSP, primeiramente utilizando um algoritmo evolutivo como gerador de soluções para o processo de agrupamento, e depois substituindo o algoritmo evolutivo por outras metaheurísticas.

Outras informações sobre os TSPP podem ser encontradas em Feillet, Dejax e Gendreau (2006), que apresentam uma pesquisa sobre os TSPP, identificando e comparando diferentes classes de aplicação, modelando algumas formulações e analisando a complexidade de algumas técnicas de solução exatas e heurísticas.

4. Clustering Search (CS)

A metaheurística *Clustering Search* (CS), proposta por Oliveira e Lorena (2004, 2007), consiste num processo de agrupamentos de soluções para detectar regiões supostamente promissoras no espaço de busca. É particularmente interessante encontrar tais regiões assim que possível para mudar a estratégia de busca sobre elas. Uma região pode ser vista como um sub-espaço de busca definido por uma relação de vizinhança.

No CS um processo de agrupamento iterativo é executado simultaneamente com uma metaheurística, identificando grupos de soluções que merecem especial interesse. As regiões destes grupos de soluções devem ser exploradas tão logo sejam detectadas, através de heurísticas de busca local específicas. Espera-se uma melhoria no processo de convergência associado a uma diminuição no esforço computacional em virtude do emprego mais racional dos métodos de busca local.

O CS procura localizar regiões promissoras através do enquadramento destas por *clusters*. Um *cluster* é definido pela tripla $\mathcal{G} = \{C; r; \beta\}$ onde C, r e β são, respectivamente, o centro e o raio de uma região de busca, e uma estratégia de busca associada ao *cluster*.

O centro C é uma solução que representa o *cluster*, identificando a sua localização dentro do espaço de busca. Inicialmente, os centros são obtidos aleatoriamente e progressivamente tendem a deslocar-se para pontos realmente promissores no espaço de busca. O raio r estabelece a distância máxima, a partir do centro, até a qual uma solução pode ser associada ao *cluster*. Por exemplo, em otimização combinatória, r pode ser definido como o número de movimentos necessários para transformar uma solução em outra. A estratégia de busca β é uma sistemática de intensificação de busca, na qual soluções de um *cluster* interagem entre si, ao longo do processo de agrupamento, gerando novas soluções.

O CS consiste em quatro componentes conceitualmente independentes com diferentes atribuições:

- uma metaheurística (ME);
- um agrupador iterativo (AI);
- um analisador de agrupamentos (AA);
- um algoritmo de otimização local (AO).

A Figura 1 ilustra os quatros componentes, o conjunto de soluções e os centros dos *clusters* que interagem entre si.

O componente ME trabalha como um gerador de soluções de tempo integral. O algoritmo é executado independentemente dos componentes restantes e este precisa ser capaz de gerar soluções de forma contínua para o processo de agrupamento. Simultaneamente, *clusters* são mantidos para representar estas soluções. Este processo trabalha como um laço infinito, no qual, soluções são geradas ao longo das iterações.

O componente AI objetiva reunir soluções similares dentro de *clusters*, mantendo uma solução no centro do *cluster* que seja representativa para as demais soluções. Para evitar um esforço computacional extra, o AI é desenvolvido como um processo *online*, no qual os agrupamentos são progressivamente alimentados por soluções geradas em cada iteração do ME. Um número máximo de *clusters* $\mathcal{MC}$ é definido a priori, evitando a criação ilimitada de *clusters*. Uma métrica de distância também precisa ser definida inicialmente, permitindo uma medida de similaridade para o processo de agrupamento. Cada solução recebida pelo AI é agrupada no *cluster* que for mais similar a ela, causando uma perturbação no centro deste. Tal perturbação é chamada de assimilação e consiste basicamente em atualizar o centro com a nova solução recebida.

O componente AA provê uma análise de cada *cluster*, em intervalos regulares, indicando um provável *cluster* promissor. A densidade do *cluster*, λ_i, é a medida que indica o nível de atividade dentro do *cluster*. Por simplicidade, λ_i pode ser o número de soluções geradas por ME e alocadas para o *cluster i*. Sempre que λ_i atinja um certo limitante, significando que algum modelo de informação foi predominantemente gerado por ME, tal *cluster* precisa ser melhor investigado para acelerar o processo de convergência deste. AA também é responsável

por eliminar os *clusters* com baixa densidade, permitindo criar novos *clusters* e preservando os *clusters* mais ativos. A eliminação do *cluster* não afeta o conjunto de soluções do ME, somente o centro do *cluster* é considerado irrelevante para o processo.

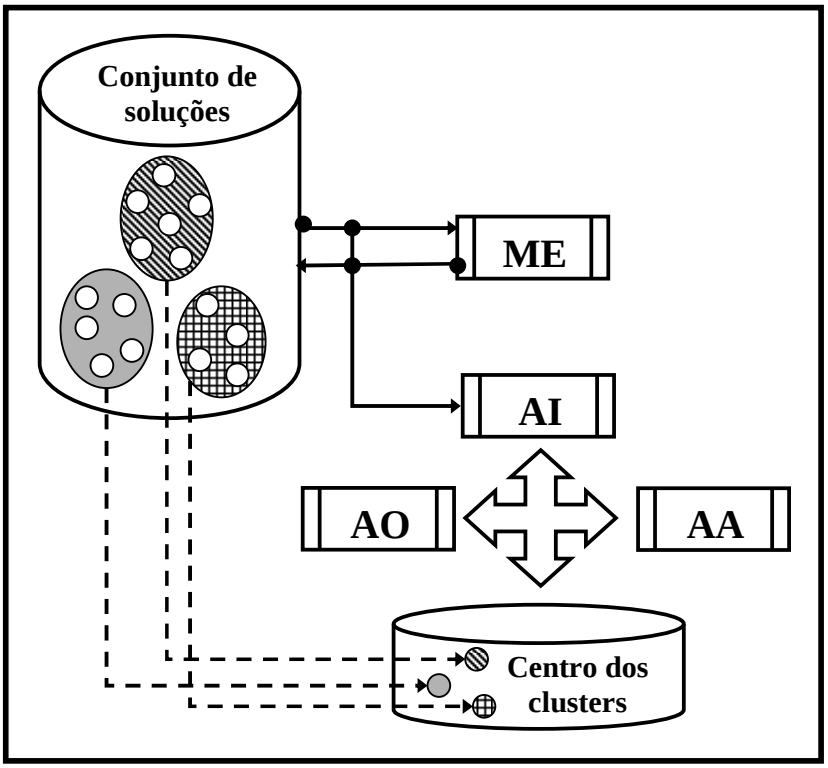

Figura 1. Diagrama Conceitual do CS

Por fim, o componente AO é um módulo de busca local que provê a exploração de uma suposta região promissora, delimitada pelo *cluster*. Este processo acontece após AA ter descoberto um *cluster* promissor e uma busca local é aplicada no centro do *cluster*. AO pode ser considerado como a estratégia de busca β associada com o *cluster*, ou seja, uma heurística específica para o problema que será empregada dentro do *cluster*.

5. CS aplicado aos TSPs with Profits

O objetivo deste trabalho é implementar um método de solução, utilizando o CS, capaz de resolver alguns dos problemas da classe TSPP, tais como, PCTSP, PTP e QTSP. Sendo que, para cada problema é necessário modificar apenas o cálculo da função objetivo.

A representação de uma solução é a mesma para todos os problemas, sendo definida através de um vetor que contém os vértices do problema na ordem em que são visitados, observando que o sinal negativo indica que o vértice não será visitado. A Figura 2 ilustra a representação de uma solução, onde a seqüência de visitas é {1,3,0,4} e os vértices 5 e 2 não são visitados.

1	**3**	**-5**	**0**	**4**	**-2**

Figura 2 – Representação de uma solução

O CS foi implementado utilizando as metaheurísticas *Greedy Randomized Adaptive Search Procedure* – GRASP (Feo e Resende, 1995) e *Variable Neighborhood Search* – VNS (Mladenovic e Hansen, 1997) como componente ME, sendo estas responsáveis por gerar soluções para o processo de agrupamento.

O GRASP é basicamente composto por duas fases: uma fase de construção, na qual

uma solução viável é gerada e uma fase de busca local, na qual a solução construída é melhorada.

Na fase de construção do GRASP utilizou-se a regra de inserção da heurística *Adding-Nodes* (Dell'Amico *et al.*, 1998) para construir a lista dos elementos candidatos (C_{el}) a serem inseridos na solução. Inicialmente dois vértices são inseridos na rota e depois cada vértice é selecionado de forma aleatória a partir de uma parte da lista C_{el}, contendo os melhores candidatos, chamada lista de candidatos restrita (LCR). Este vértice é inserido na solução e a lista de candidatos é atualizada a cada iteração. A fase de construção termina quando não houver candidatos com economia positiva e o prêmio mínimo tenha sido coletado. A regra de inserção para adicionar o vértice v_k entre os vértices v_i e v_j é

$$h(v_k) = c_{v_i v_j} + \gamma_{v_k} - c_{v_i v_k} - c_{v_k v_j} \quad (1)$$

que é composta pelo custo da aresta (v_i,v_j), a penalidade do vértice v_k e os custos das arestas (v_i,v_k) e (v_k,v_j), respectivamente.

Um parâmetro $\alpha \in [0,1]$ determina o tamanho da LCR. Valores de α próximos a zero conduzem a soluções muito próximas àquela obtida escolhendo-se a cada passo o melhor elemento segundo a função h, enquanto que valores de α próximo a 1 conduzem a soluções praticamente aleatórias, o que pode tornar o processo de busca local mais lento.

Na fase de busca local do GRASP será utilizado a metaheurística VNS procurando refinar a solução construída. Sendo que este explora o espaço de busca através de trocas sistemáticas de vizinhanças, aplicando um método de busca local sobre o vizinho gerado em uma determinada vizinhança.

As estruturas de vizinhanças são definidas através de movimentos aleatórios. No VNS proposto definem-se nove estruturas de vizinhança aninhadas, através dos seguintes movimentos:

- m_1: Inserir 1 vértice que não está sendo visitado;
- m_2: Retirar 1 vértice que está sendo visitado;
- m_3: Trocar 2 vértices que estão sendo visitados de posição;
- m_4: Inserir 2 vértices que não estão sendo visitados;
- m_5: Retirar 2 vértices que estão sendo visitados;
- m_6: Trocar 4 vértices que estão sendo visitados de posição;
- m_7: Inserir 3 vértices que não estão sendo visitados;
- m_8: Retirar 3 vértices que estão sendo visitados;
- m_9: Trocar 6 vértices que estão sendo visitados de posição.

A partir da solução construída, a cada iteração seleciona-se aleatoriamente um vizinho s' na k-ésima vizinhança da solução corrente s, realizando-se o movimento m_k definido anteriormente. Esse vizinho é então submetido a um procedimento de busca local. Se o ótimo local, s'', for melhor que a solução s corrente, a busca continua de s'' recomeçando da primeira estrutura de vizinhança. Caso contrário, a busca continua a partir da próxima vizinhança. Este procedimento é encerrado quando o número de iterações sem melhora na solução corrente for maior que 1000.

Como um ótimo local em uma vizinhança não é necessariamente o mesmo segundo outra vizinhança, mudanças de vizinhanças também podem ser executadas na fase de busca local do VNS. Este método de busca local é chamado *Variable Neighborhood Descent* (VND) (Mladenovic e Hansen, 1997) e será utilizado para refinar o vizinho gerado no VNS.

O VND foi implementado como uma busca local do VNS, sendo composto por três diferentes heurísticas de refinamento, que combinam dois movimentos: *Add-step* e *Drop-step* (Gomes *et al.* 2000). O movimento *Add-step* consiste em adicionar o vértice que possuir o melhor valor de economia de inserção (Figura 3). O movimento *Drop-step* consiste em retirar o vértice que possuir o melhor valor de economia de remoção (Figura 4). Em ambos movimentos, se o valor da economia for positivo então a função objetivo irá melhorar após o movimento. O principal aspecto a ser observado é que todos os movimentos são executados preservando a viabilidade da solução.

As heurísticas de refinamento do VND são:

- ***SeqDrop***: aplicar uma seqüência de movimentos *Drop-step*, enquanto o

valor da função objetivo estiver melhorando;

- ***Add-Drop***: aplicar um movimento *Add-step* e um movimento *Drop-step*;
- ***SeqAdd***: aplicar uma seqüência de movimentos *Add-step* enquanto o valor da função objetivo estiver melhorando.

procedimento Add-step

para (cada $v_k \in$ R) **faça**

$$L \leftarrow z(v_k) = \max_{(v_i, v_j) \in A(R)} \{c_{v_i v_j} + \gamma_{v_k} - c_{v_i v_k} - c_{v_k v_j}\}$$

Selecionar $v_k = \max\{z(v_k) \text{ para cada } k \in L\}$

Inserir v_k na rota R

fim procedimento

Figura 3. Movimento *Add-step*

procedimento Drop-step

para (cada $v_k \in$ R) **faça**

$$L \leftarrow t(v_k) = c_{v_i v_k} + c_{v_k v_j} - \gamma_{v_k} - c_{v_i v_j}\}$$

Selecionar $v_k = \max\{t(v_k) \text{ para cada } k \in L\}$

Remover v_k da rota R

fim procedimento

Figura 4. Movimento *Drop-step*

Toda vez que alguma heurística encontrar uma solução melhor que a solução corrente, o VND retorna para a primeira heurística. A condição de parada do VND é não existir mais melhoras para a solução através destas heurísticas.

O componente AI realiza um agrupamento iterativo de cada solução vizinha obtida pelo GRASP/VNS após ser aplicado o VND. Define-se inicialmente um número máximo de *clusters* $\mathcal{MC}$, objetivando evitar que o processo de agrupamento se torne muito lento. O *i*-ésimo *cluster* tem o seu próprio centro C_i e um raio *r* que é comum aos demais *clusters*.

Se a distância métrica entre a solução GRASP/VNS e o centro de algum *cluster* for menor que o raio deste, a informação da solução deve causar uma perturbação (assimilação) no centro do *cluster* mais similar. Caso contrário, a informação da solução é considerada nova e esta é armazenada em um novo *cluster* se houver um número de *clusters* menor que $\mathcal{MC}$. A medida de distância utilizada neste trabalho será o número de arestas diferentes entre a solução e o centro do *cluster*, sendo assim, quanto maior o número de arestas diferentes entre duas soluções, maior será a distância entre elas e menor será a similaridade.

No processo de assimilação será utilizado o método *path-relinking* (Glover, 1996), o qual realiza movimentos exploratórios na trajetória que interconecta a solução gerada pelo GRASP/VNS (s_k) e o centro do *cluster* mais similar (C_i). Assim sendo, o próprio processo de assimilação já é uma forma de busca local dentro do *cluster*, pois o centro vai ser deslocado para a melhor solução avaliada nessa trajetória, caso ela seja melhor que o centro do *cluster*. A Figura 5 apresenta um exemplo do *path-relinking* aplicado aos TSPP.

O componente AA é executado toda vez que uma solução for atribuído a um *cluster*. A função do AA é verificar se o *cluster* já pode ser considerado promissor. Um *cluster* se torna promissor quando atinge uma certa densidade λ dada por,

$$\lambda = PD.\left(\frac{NS}{|Clus|}\right) \tag{18}$$

onde, *PD* é a pressão de densidade que permite controlar a sensibilidade do componente AA. Esse parâmetro indica quantas vezes a densidade deve estar acima do normal para que um *cluster*

seja considerado promissor. A densidade normal é obtida se *NS* (número de soluções geradas em cada intervalo de análise dos *clusters*) fosse igualmente dividido por todos *clusters* existentes (|*Clus*|). Caso a densidade do *cluster* seja maior que esse valor significa que um certo padrão de informação se tornou predominante no processo de busca, sendo o centro deste *cluster* explorado através do componente AO.

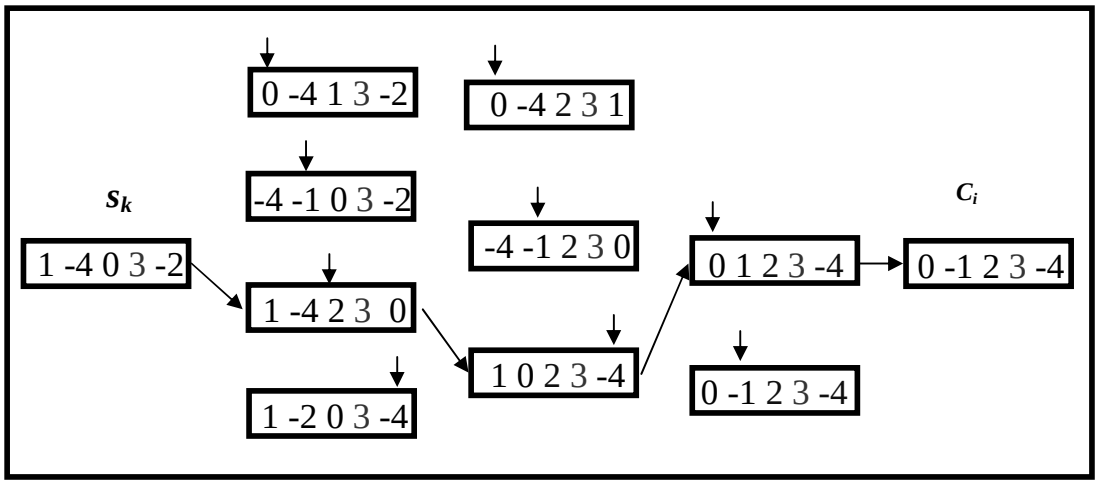

Figura 5. Exemplo do *path-relinking* aplicado aos TSPP.

O componente AA também tem como função executar um esfriamento de todos os *clusters* que são explorados pelo AO, ou seja, diminuir a densidade destes *clusters*. Outra função do AA é eliminar, a cada intervalo de análise, os *clusters* com baixa densidade.

No componente AO foi implementado a heurística 2-Opt (Croes, 1958), objetivando melhorar a solução presente no centro de um *cluster* promissor. Este método examina todas as possíveis trocas entre pares de arestas não consecutivas, realizando a que fornecer o maior ganho em termos de melhora da função objetivo. Sendo executado enquanto houver alguma troca de duas arestas que melhore a solução. Caso AO encontre uma solução que seja melhor que o centro do *cluster*, este é atualizado com a solução encontrada.

A Figura 6 apresenta o pseudo-código da abordagem CS. Este procedimento é executado enquanto o número de iterações não for satisfeito (neste trabalho foram utilizadas 4000 iterações). Na fase de construção do GRASP uma solução inicial é iterativamente construída, sendo esta solução refinada na fase de busca local por meio do VNS. A cada iteração do VNS um vizinho é gerado aleatoriamente e aplica-se o método VND neste vizinho. Toda solução encontrada pelo VND é agrupada pelo componente AI em algum *cluster*, sendo este analisado pelo componente AA. Caso o *cluster* seja considerado promissor, executa-se o componente AO.

6. Resultados Computacionais

A abordagem CS para os TSPP foi codificada em C++ e os experimentos foram conduzidos em uma máquina Pentium 4 3.02 GHz com 512 MB de memória. Os experimentos foram realizados com objetivo de validar a abordagem proposta, mostrando que algoritmos de busca por agrupamentos podem ser competitivos para resolução dos problemas da classe TSPP.

Os TSPP não possuem uma biblioteca pública de problemas-teste, sendo assim, Chaves *et al.* (2004) geraram aleatoriamente um conjunto de problemas-teste para o PCTSP com $n \in \{11, 21, 31, 51, 101, 251, 501\}$ e os seguintes intervalos:

- custo de deslocamento entre os vértices: $c_{v_i v_j} \in [50,1000]$;
- prêmio associado à cada vértice: $p_{v_i} \in [1,100]$;
- penalidade associada à cada vértice: $\gamma_{v_i} \in [1,750]$.

```
procedimento CS
    para (número de iterações não satisfeito) faça
        s = ∅
        enquanto (solução não construída) faça
            produzir Lista de Candidatos (C_el)
            LCR = C_el * α
            e = selecionar elemento aleatoriamente(LCR)
            s = s ∪ {e}
            atualizar lista de Candidatos(C)
        fim enquanto
        k_max = número de vizinhanças
        enquanto (critério de parada não satisfeito ) faça
            k ← 1
            enquanto (k ≤ k_max)
                gerar um vizinho s'qualquer na vizinhança N^k(s)
                s" = Aplicar VND em s'
                aplicar o componente AI (s")
                aplicar o componente AA (cluster ativo)
                se (cluster ativo for promissor) faça
                    aplicar o componente AO
                se (s" for melhor que s) faça
                    s ← s"
                    k ← 1
                senão
                    k ← k + 1
            fim enquanto
        fim enquanto
    fim para
fim procedimento CS
```

Figura 6 – Pseudo-código do CS

Para o PTP não são considerados os prêmios associados aos vértices, enquanto que o QTSP não leva em consideração as penalidades associadas aos vértices. O valor do prêmio mínimo, p_{min}, a ser coletado no PCTSP e no QTSP representa 75% do somatório de todos os prêmios associados aos vértices. Este conjunto de problemas testes estão disponíveis em http://www.lac.inpe.br/\~lorena/intancias.html.

Os valores para os parâmetros de desempenho da abordagem CS foram ajustados através de várias execuções e também baseados em Oliveira e Lorena (2004).

- número de soluções geradas no intervalo de cada análise *NS* = 200;
- número máximo de *clusters* $\mathcal{MC}$ = 20;
- pressão de densidade *PD* = 2,5;
- parâmetro α = 0,2;

Para validar os resultados obtidos pelas abordagens heurísticas, foi proposto resolver as formulações matemáticas dos TSPP, apresentada em Chaves e Lorena (2005), através do *software* CPLEX versão 10.0.1 (Ilog, 2001). Porém, devido à complexidade dos problemas, o CPLEX consegue resolver apenas problemas-teste de pequeno porte. Observa-se através dos resultados das Tabelas 1, 2 e 3 que para problemas-teste com 101 vértices o CPLEX levou várias horas para obter a solução ótima e para os problemas-teste maiores o CPLEX não consegue encontrar nenhuma solução factível em 200.000 segundos de execução.

As Tabelas a seguir apresentam os resultados encontrados pelo CS e pelo GRASP/VNS sem o processo de agrupamento, além dos resultados obtidos pelo CPLEX para o PCTSP, QTSP e PTP. A melhor solução encontrada (MS), a solução média (SM) e o tempo médio de execução

para encontrar a melhor solução (TE), em segundos, são considerados para comparar as performances das abordagens. Os valores em negrito indicam qual abordagem encontrou os melhores valores de função objetivo e tempo de execução para cada problema-teste.

A Tabela 1 apresenta os resultados computacionais encontrados para o PCTSP. Pode-se observar que, CS obteve os melhores resultados para quase todos os problemas-teste, encontrando o ótimo global para problemas com até 51 vértices e soluções próximas do ótimo para os problemas-teste *v100a* e *v100b*. Para os problemas-teste maiores, o CS conseguiu encontrar soluções viáveis em um tempo computacional razoável. A abordagem GRASP/VNS não conseguiu encontrar o ótimo global para problemas menores, e para os problemas maiores os resultados foram bem piores que os do CS. Outra vantagem do CS é o fato da solução média está próxima da melhor solução encontrada.

		CPLEX		**CS**			**GRASP/VNS**		
Prob.	***n***	**MS**	**TE (s)**	**MS**	**SM**	**TE (s)**	**MS**	**SM**	**TE (s)**
v10	11	1765	0,11	**1765**	**1765**	**0,02**	1765	1765	0,05
v20	21	2302	2,65	**2302**	**2302**	**0,35**	2355	2403	0,27
v30a	31	3582	4,89	**3582**	**3582**	**0,82**	3734	3899	6,00
v30b	31	2515	6,18	**2515**	**2515**	**3,14**	2647	2771	5,93
v30c	31	3236	20,24	**3236**	**3242**	**1,54**	3301	3356	6,04
v50a	51	4328	1032,79	**4328**	**4335**	**9,78**	4752	4900	13,41
v50b	51	3872	634,01	**3872**	**3878**	**6,40**	4097	4383	35,80
v100a	101	**6762**	**86390,66**	6785	6822	108,55	8302	8476	52,69
v100b	101	**6760**	**85002,51**	6782	6844	216,16	7890	8367	87,78
v250a	251	-	-	**14483**	**14608**	**713,12**	18306	18510	364,16
v250b	251	-	-	**14078**	**14176**	**701,58**	17871	18191	393,47
v500a	501	-	-	**27189**	**27230**	**2880,03**	34437	34771	463,02
v500b	501	-	-	**28133**	**28334**	**2080,89**	34841	35296	529,72

Tabela 1 – Resultados computacionais para o PCTSP.

A Tabela 2 apresenta os resultados para o QTSP. O CS mais uma vez encontrou os melhores resultados, obtendo a solução ótima ou soluções próximas ao ótimo para problemas-teste onde estes são conhecidos. O GRASP/VNS não obteve bons resultados, encontrando o ótimo global somente para problemas com até 11 e 21 vértices, e ficando consideravelmente longe das soluções encontradas pelo CS para os problemas maiores.

		CPLEX		**CS**			**GRASP/VNS**		
Prob.	***n***	**MS**	**TE (s)**	**MS**	**SM**	**TE (s)**	**MS**	**SM**	**TE (s)**
v10	11	1755	0,96	**1755**	**1755**	**0,01**	1755	1755	0,07
v20	21	1439	5,13	**1439**	**1439**	**1,33**	1439	1469	1,24
v30a	31	2074	29,56	**2074**	**2075**	**3,09**	2182	2268	5,65
v30b	31	1552	15,40	**1552**	**1552**	**6,62**	1553	1772	4,01
v30c	31	1966	30,23	**1966**	**2004**	**2,49**	2086	2177	6,20
v50a	51	2438	372,09	**2438**	**2466**	**17,76**	2912	2956	11,31
v50b	51	2192	1231,20	**2192**	**2218**	**35,90**	2548	2600	24,06
v100a	101	**3770**	**46331,41**	3862	3930	84,94	4777	4811	68,15
v100b	101	**3935**	**45615,01**	4030	4092	126,07	5092	5191	44,58
v250a	251	-	-	**8253**	**8342**	**518,82**	10309	10626	295,34
v250b	251	-	-	**8336**	**8508**	**549,76**	10701	10865	313,50
v500a	501	-	-	**15367**	**15780**	**2207,11**	19510	19717	799,72
v500b	501	-	-	**15211**	**15757**	**1526,38**	19494	19567	750,24

Tabela 2 – Resultados computacionais para o QTSP.

A Tabela 3 apresenta os resultados para o PTP, sendo que, o CS novamente obteve os melhores resultados. Através desta Tabela podemos observar que o CS obteve a solução ótima para problemas-teste com até 51 vértices, enquanto que o GRASP/VNS encontrou o ótimo somente para os problemas-teste menores. É importante ressaltar também que ambas abordagens conseguiram encontrar soluções viáveis em um tempo de execução razoável para os problemas-teste maiores, porém as soluções do CS são melhores.

		CPLEX		**CS**			**GRASP/VNS**		
Prob.	***n***	**MS**	**TE (s)**	**MS**	**SM**	**TE (s)**	**MS**	**SM**	**TE (s)**
v10	11	1558	0,29	**1558**	**1558**	**0,001**	1558	1558	0,001
v20	21	2302	2,22	**2302**	**2302**	**0,94**	2302	2369	2,24
v30a	31	3582	3,94	**3582**	**3582**	**1,89**	3695	3871	9,48
v30b	31	2515	6,08	**2515**	**2515**	**1,92**	2571	2617	10,09
v30c	31	3236	16,30	**3236**	**3242**	**2,22**	3345	3408	6,55
v50a	51	4328	1261,03	**4328**	**4338**	**22,97**	4739	4852	19,33
v50b	51	3872	910,37	**3872**	**3877**	**12,62**	4219	4371	18,51
v100a	101	**6762**	**197783,19**	6784	6799	125,96	8191	8344	69,78
v100b	101	**6760**	**55953,18**	6826	6864	150,05	8212	8342	55,91
v250a	251	-	-	**14493**	**14584**	**1124,62**	17970	18432	733,19
v250b	251	-	-	**14055**	**14119**	**901,94**	17302	17786	637,00
v500a	501	-	-	**27316**	**27564**	**2387,02**	33727	34157	1401,86
v500b	501	-	-	**27890**	**27944**	**2345,02**	34757	35044	1220,97

Tabela 3 – Resultados computacionais para o PTP.

7. Conclusão

Este trabalho propõe uma abordagem heurística, utilizando a metaheurística *Clustering Search*, para a resolução de alguns problemas da classe *TSPs with Profits*. Esta utiliza o conceito de algoritmos híbridos, combinando metaheurísticas com um processo de agrupamento de soluções em sub-espaços de busca (*clusters*), visando detectar regiões promissoras. Sempre que uma região for considerada promissora é realizada uma intensificação da busca nesta região, objetivando uma aplicação mais racional do método de busca local.

Este trabalho se justifica segundo alguns aspectos. Primeiramente, pelo fato do CS ser um método novo e que vem sendo aplicado a problemas de otimização combinatória encontrados na literatura, tais como, seqüênciamento de padrões e minimização de funções numéricas.

Além disso, os resultados obtidos mostram que a abordagem CS é competitiva para resolução do PCTSP, PTP e QTSP, conseguindo encontrar o ótimo global para problemas-teste com até 51 vértices em um tempo de execução razoável, sendo necessário modificar somente a função objetivo do problema a ser tratado pelo CS.

Tópicos interessantes para pesquisas futuras na área incluem estudos sobre a robustez do método aqui proposto, diante de mudanças nos parâmetros dos problemas-teste (por exemplo, no grau de esparsidade dos grafos, na variabilidade dos valores aleatoriamente gerados, na percentagem de prêmio mínimo a ser coletado, etc) lidando com a conseqüente ampliação do número de experimentos via análise estatística dos resultados. Também é de interesse avaliar a qualidade das soluções obtidas em grafos realísticos (por exemplo, planares), bem como em grafos adaptados de problemas-teste criados para o caixeiro viajante original e suas variantes, disponíveis em bibliotecas consagradas (Beasley, 1990; Reineilt).

Referências

Awerbuch, B.; Azar, Y.; Blum, A.; Vempala, S. (1998), New approximation guarantees for minimum weight k-trees and prize-collecting salesmen. *SIAM Journal on Computing*, v. 28, n. 1, p. 254–262.

Balas, E. (1989), The prize collecting traveling salesman problem. *Networks*, v. 19, p. 621–636.

Beasley, J. E. (1990), Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, v. 41, p. 1069–1072.

Chaves, A.A.; Biajoli, F.L.; Mine, O.M.; Souza, M.J.F. (2004), Modelagens exata e heurística para resolução de uma generalização do problema do caixeiro viajante. *Simpósio Brasileiro de Pesquisa Operacional*, 36, p. 1367–1378.

Chaves, A.A.; Lorena, L.A.N. (2005), Hybrid algorithms with detection of promising areas for the prize collecting traveling salesman problem. International Conference on Hybrid Intelligent Systems, 5, Proceedings... Los Alamitos, California: IEEE Computer Society, p. 49–54.

Croes, G. (1958), A method for solving traveling salesman problems. *Operations Research*, v. 6, p. 791–812.

Dell'Amico, M.; Maffioli, F.; Sciomanchen, A. (1998), A lagrangian heuristic for the prize collecting traveling salesman problem. *Annals of Operations Research*, v. 81, p. 289–305.

Dell'Amico, M.; Maffioli, F.; Värbrand, P. (1995), On prize collecting tours and the asymetric traveling salesman problem. *International Transactions in Operational Research*, v. 2, n. 3, p. 297–308.

Feillet, D.; Dejax, P.; Gendreau, M. (2005), Traveling salesman problems with profits. *Transportation Science*, v. 2, n. 39, p. 188–205.

Feo, T.A.; Resende, M.G.C. (1995), Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109–133.

Glover, F. (1996), Tabu search and adaptive memory programming: Advances, applications and challenges. Em: Interfaces in Computer Science and Operations Research (Eds. Barr, R.S.; Helgason., R.V.; Kennington, J.L.), Kluwer, p. 1–75.

Gomes, L.M.; Diniz, V.B.; Martinhon, C.A. (2000), An hybrid grasp+vnd metaheuristic for the prize collecting traveling salesman problem. *Simpósio Brasileiro de Pesquisa Operacional*, 32, p. 1657–1665.

ILOG. ILOG CPLEX 7.5 *Reference Manual*. France, 2001. 610 p.

Keller, C.P.; Goodchild, M. (1988), The multiobjective vending problem: A generalization of the traveling salesman problem. *Environment and Planning B: Planning and Design*, v. 15, p. 447–460.

Laporte, G.; Martello, S. (1990), The selective traveling salesman problem. *Discrete Applied Mathematics*, v. 26, p. 193–207.

Mladenovic, N.; Hansen, P. (1997), Variable neighborhood search. *Computers and Operations Research*, v. 24, p. 1097–1100.

Oliveira, A.C.M.; Lorena, L.A.N. (2004), Detecting promising areas by evolutionary clustering search. Em: Advances in Artificial Intelligence (Eds. Bazzan, A.L.C.; Labidi, S.). Springer Lecture Notes in Artificial Intelligence Series, p. 385–394.

Oliveira, A.C.M. and Lorena, L.A.N. (2007), Hybrid Evolutionary Algorithms and Clustering Search. Em: Springer SCI Series (Eds: Crina Grosan, Ajith Abraham and Hisao Ishibuchi), – aceito para publicação (http://www.lac.inpe.br/~lorena/alexandre/HEA-07.pdf).

Reinelt, G., *Traveling Salesman : Computational Solutions for TSP Applications*. Springer: Lecture Notes in Computer Science, 1994.

Reinelt, G., TSPLIB. Disponível em: (http://www.iwr.uni-heidelberg.de/groups/comopt/software/ TSPLIB95). Acesso em: 20/03/2007.

Tsiligirides, T. (1984), Heuristic methods applied to orienteering. *Journal of Operational Research Society*, v. 35, n. 9, p. 797–809.