MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE

HEURÍSTICAS HÍBRIDAS COM BUSCA ATRAVÉS DE AGRUPAMENTOS PARA O PROBLEMA DO CAIXEIRO VIAJANTE COM COLETA DE PRÊMIOS

Antonio Augusto Chaves

Proposta de Mestrado do Curso de Pós-graduação em Computação Aplicada, orientada pelo Dr. Luiz Antonio Nogueira Lorena, em 30 de maio de 2005.

INPE
São José dos Campos
2005

...(...)

CHAVES, A. A.

Heurísticas Híbridas com Busca através de Agrupamentos para o Problema do Caixeiro Viajante com Coleta de Prêmios / A. A. Chaves. – São José dos Campos: INPE, 2005.

..... – (INPE-....-.../...).

1. Problema do Caixeiro Viajante. 2. Heurísticas. 3. *Evolutionary Clustering Search* 4. Algoritmo de Treinamento Populacional. 5. GRASP. 6. VNS 7. Programação Linear.

RESUMO

O Problema do Caixeiro Viajante com Coleta de Prêmios (PCVCP) é uma generalização do Problema do Caixeiro Viajante, pode ser associado a um caixeiro que coleta um prêmio p_k, não negativo, em cada cidade k visitada e paga uma penalidade γ_t para cada cidade t não visitada, com um custo c_{ij} de deslocamento entre as cidades i e j. O objetivo é minimizar o somatório dos custos da viagem e penalidades, incluindo na rota um número suficiente de cidades que permitam coletar um prêmio mínimo, p_{min}, pré-estabelecido. Esta proposta aborda novas técnicas heurísticas para resolver o PCVCP, utilizando um algoritmo evolutivo híbrido, chamado *Evolutionary Clustering Search* (ECS) e uma adaptação deste, chamada *CS. A validação das soluções obtidas se dará através da comparação com os resultados encontrados através de algoritmos exatos (para instâncias pequenas) e heurísticas, existentes na literatura, para instâncias maiores.

HYBRID HEURISTICS TO THE PRIZE-COLLECTING TRAVELING SALESMAN PROBLEM

ABSTRACT

The Prize Collecting Travelling Salesman Problem (PCTSP) is a generalization of the Travelling Salesman Problem, it can be associated a salesman that collects a prize p_k, no negative, in each visited city k and pays a penalty γ_t for each city t no visited, with a cost c_{ij} among the cities i and j. The objective is to minimize the sum of the costs of the trip and penalties, including in the route an enough number of cities that allow to collect a minimum prize, p_{min}. This proposal approaches new heuristic techniques to solve the PCTSP, using a hybrid evolutionary algorithm, called Evolutionary Clustering Search (ECS) and an adaptation of this, called *CS. The validation of the obtained solutions will be through the comparison with the results found through exact algorithms (for small instances) and heuristics, existent in the literature, for larger instances.

SUMÁRIO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE SÍMBOLOS

n	–	Número de vértices do problema
p_k	–	Prêmio associado ao vértice k
γ_t	–	Penalidade associada ao vértice t
c_{ij}	–	Custo de deslocamento entre os vértices i e j
p_{min}	–	Prêmio mínimo a ser coletado
α	–	Controla quão gulosa será uma solução construída no GRASP
N_k	–	Representação da k-ésima vizinhança do VNS
f	–	Função que avalia a qualidade de um indivíduo
g	–	Função heurística
δ	–	*Ranking* de um indivíduo
G_{max}	–	Valor máximo que as funções f e g podem assumir
d	–	Constante de equilíbrio do *ranking*
τ	–	Limiar de rejeição adaptativo
ξ	–	Incremento do limiar de rejeição adaptativo
$\|P\|$	–	Tamanho da população corrente
$\|P_0\|$	–	Tamanho da população inicial
s_k	–	Indivíduo
s_{base}, s_{guia}	–	Indivíduos base e guia, usados na seleção *base-guia*
c_i	–	Centro do *cluster* i
r	–	Raio dos *clusters*
β	–	Estratégia de busca associada aos *clusters*
NS	–	Número de indivíduos selecionados ou atualizados
$\|C_t\|$	–	Número de *clusters* em uma geração t
MC	–	Número máximo de *clusters*
$MAX_{geracoes}$	–	Número máximo de gerações no ATP
$\rho(e)$	–	Percentual da população considerada base (elite)
$\rho(m)$	–	Probabilidade de mutação
$\rho(\alpha)$	–	Probabilidade de escolha do α_i
λ_t	–	Densidade do *cluster* t
PD	–	Pressão de densidade
ψ	–	Conjunto discreto com os valores possíveis de α
z^*	–	Melhor solução encontrada na fase de constução GRASP
A_i	–	Valor médio das soluções obtidas na fase de construção GRASP
s'	–	Solução vizinha da solução corrente s
s''	–	Ótimo local obtido a partir do visinho s'
$NIter$	–	Número de iterações do GRASP Reativo

LISTA DE SIGLAS E ABREVIATURAS

PCVCP	–	Problema do Caixeiro Viajante com Coleta de Prêmios
AGs	–	Algoritmos Genéticos
ECS	–	*Evolutionary Clustering Search*
*CS	–	*Clustering Search*, associado a uma metaheurística qualquer
AGC	–	Algoritmo Genético Construtivo
ATP	–	Algoritmo de Treinamento Populacional
GRASP	–	*Greedy Randomized Adaptive Search Procedure*
VNS	–	*Variable Neighborhood Search*
PCV	–	Problema do Caixeiro Viajante
PL	–	Programação Linear
PLI	–	Programação Linear Inteira
C	–	Lista de candidados da fase de construção GRASP
LCR	–	Lista de candidados restrita da fase de construção GRASP
AE	–	Algoritmo evolutivo
AI	–	Agrupador iterativo
AA	–	Analisador de agrupamentos
AO	–	Algoritmo de otimização local

CAPÍTULO 1

INTRODUÇÃO

Este trabalho tem como objetivo abordar novas técnicas heurísticas para resolver o Problema do Caixeiro Viajante com Coleta de Prêmios (PCVCP), referido na literatura como *Prize Collecting Traveling Salesman Problem*. Sendo uma generalização do Problema do Caixeiro Viajante, pode ser associado a um caixeiro que coleta um prêmio p_k, não negativo, em cada cidade k que ele visita e paga uma penalidade γ_t para cada cidade t que não visita, com um custo c_{ij} de deslocamento entre as cidades i e j. O problema encontra-se em minimizar o somatório dos custos da viagem e penalidades, enquanto inclui na sua rota um número suficiente de cidades que permitam coletar um prêmio mínimo, p_{min}, pré-estabelecido.

A escolha deste problema para estudo foi feita em função da fácil adaptação deste a situações da vida real. Podendo ser descrito como um universo de clientes em potencial, onde existe associado a cada cliente, quando não for atendido, uma penalidade pela expectativa de atendimento ou importância, e quando este for atendido, um ganho relativo. Deseja-se a partir de uma origem, montar um percurso contendo alguns clientes visitados uma única vez retornando ao ponto de partida, minimizando o custo da distância total percorrida e a soma das penalidades, de forma a garantir um ganho mínimo que justifique o investimento.

A dificuldade de solução do PCVCP está no número elevado de soluções existentes. Assumindo que o custo de deslocamento de uma cidade i a outra j seja simétrica, isto é, que $c_{ij} = c_{ji}$, o número total de soluções possíveis é (n - 1)! / 2, sendo classificado na literatura como $NP-$*difícil*, como apresentado em (Fischetti e Toth, 1988), isto é, não existem algoritmos que o resolvam em tempo polinomial. Mesmo com os rápidos avanços tecnológicos dos computadores, uma enumeração completa de todas essas soluções é inconcebível para valores elevados de n. Para se resolver o PCVCP por enumeração, para $n = 20$, tem-se 6×10^{16} rotas possíveis, desta forma, um computador que avalia uma rota em cerca de 10^{-8} segundos, gastaria 19 anos para encontrar a melhor rota.

Problemas desta natureza são comumente abordados através de heurísticas. Define-se heurística como sendo uma técnica que procura boas soluções (próximas da otimalidade) a um custo computacional razoável, sem, no entanto, estar capacitada a garantir a otimalidade, bem como garantir quão próxima uma determinada solução está da solução ótima. A grande desvantagem das heurísticas reside na dificuldade de fugir de ótimos locais, o que deu origem à outra metodologia, chamada de metaheurística, que possui

ferramentas que possibilitam sair destes ótimos locais, permitindo a busca em regiões mais promissoras. O grande desafio é produzir, em tempo razoável, soluções tão próximas quanto possíveis da solução ótima.

Dentre os procedimentos enquadrados como metaheurísticas que surgiram ao longo das últimas décadas, destacam-se: Algoritmos Genéticos (AGs) (Goldberg, 1989), *Simulated Annealing* (Kirkpatrick et al., 1983), Busca Tabu (Glover, 1986), *Greedy Randomized Adaptive Search Procedure* (GRASP) (Feo e Resende, 1995), Colônia de Formigas (Dorigo et al., 1996), *Variable Neighborhood Search* (VNS) (Mladenovic e Hansen, 1997), entre outros.

Num problema típico de minimização, encontra-se um ótimo local quando qualquer movimento a ser feito piore o valor atual da função objetivo. Um ótimo global corresponde ao menor valor da função objetivo, entre todos os ótimos locais existentes no espaço de busca. A figura 1.1 ilustra o espaço de busca para um problema de minimização.

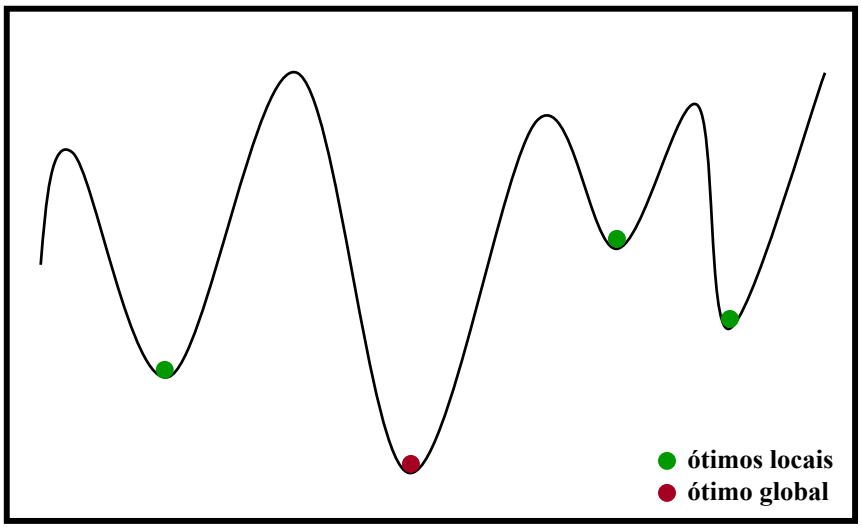

FIGURA 1.1 – Representação gráfica de um problema de minimização.

Para a resolução do PCVCP, faz-se uso de conceitos de uma técnica recente, proposta por Oliveira (2004), chamada *Evolutionary Clustering Search* (ECS) que consiste em combinar Algoritmos Evolutivos, com um método de agrupamento dos indivíduos de uma população e um método de busca local a ser realizada nos grupos de indivíduos mais promissores. Espera-se com isso uma melhoria na velocidade de convergência, como também uma diminuição do esforço computacional, uma vez que a aplicação da busca local será realizada de forma mais racional. Verificaremos também o comportamento deste método substituindo o algoritmo evolutivo por outras metaheurísticas, tais como GRASP (Feo e Resende, 1995) e VNS (Mladenovic e Hansen, 1997).

De forma a validar as soluções obtidas, propõe-se uma formulação matemática baseada em Balas (1989) e Torres e Brito (2003). Utiliza-se o *software* CPLEX (ILOG, 2001) para resolver esta formulação para instâncias de pequeno porte.

O restante do trabalho está organizado da seguinte forma. O Capítulo 2 faz uma descrição sobre o PCVCP. O Capítulo 3 faz uma breve revisão bibliográfica sobre o PCVCP. No Capítulo 4 são descritas as técnicas que serão utilizadas neste trabalho. No Capítulo 5 são descritas, em detalhes, as abordagens propostas para resolver o PCVCP. O Capítulo 6 apresenta alguns resultados preliminares obtidos até o presente momento pelas abordagens propostas. E, no Capítulo 7, são descritas algumas considerações a respeito dessas abordagens e do trabalho como um todo.

CAPÍTULO 2

DESCRIÇÃO DO PROBLEMA

O PCVCP foi formulado inicialmente por Egon Balas, (Balas, 1989), como um modelo para a programação da operação diária de uma fábrica que produzia lâminas de aço. Por razões que tinham a ver com o desgaste dos rolos e também por outros fatores, a seqüência na ordem do processamento era essencial. A programação consistia na escolha de um número de lâminas associadas às suas ordens de execução, que satisfizessem o limite inferior do peso, e que ordenadas numa seqüência apropriada, minimizasse a função de seqüência. As tarefas de escolha das lâminas e das opções disponíveis para o seu sequenciamento, necessitavam ser resolvidas em conjunto. Chamado então de *Prize Collecting Traveling Salesman Problem*, ou seja, Problema do Caixeiro Viajante com Coleta de Prêmios, serviu como base para o desenvolvimento de um software implementado por Balas e Martin (1985), que utilizava a combinação de várias heurísticas para encontrar boas soluções, organizando-as em programações diárias.

O problema do caixeiro viajante (PCV) (Reinelt, 1994) é um dos mais tradicionais e conhecidos problemas de otimização combinatótia. Os problemas de roteamento lidam em sua maior parte com rotas sobre pontos de demanda ou oferta. Esses pontos podem ser representados por cidades, postos de trabalho ou atendimento, clientes, depósitos, etc. O PCV é descrito por um conjunto de n cidades e uma matriz de distância entre elas, tendo o seguinte objetivo: o caixeiro viajante deve sair de uma cidade chamada origem, visitar cada uma das n - 1 cidades restantes apenas uma única vez e retornar à cidade origem percorrendo a menor distância possível, ou seja, deve ser encontrada uma rota fechada (ciclo hamiltoniano) de comprimento mínimo que passe exatamente uma única vez por cada cidade.

Um grande número de problemas de roteamento e planejamento podem ser formulados como uma generalização do Problema do Caixeiro Viajante. Neste caso, o PCVCP pode ser associado a um caixeiro viajante que coleta um prêmio p_k, não negativo, em cada cidade k que ele visita e paga uma penalidade γ_t para cada cidade t que não visita, com um custo c_{ij} de deslocamento entre as cidades i e j. O problema encontra-se em minimizar o somatório dos custos da viagem e penalidades, enquanto inclui na sua rota um número suficiente de cidades que permitam coletar um prêmio mínimo, p_{min}, pré-estabelecido.

O PCVCP pode ser representado conforme a figura 2.1, sendo $G = (V, E)$ um grafo completo não direcionado com um conjunto de vértices V e um conjunto de arestas E. Associado com cada aresta $e = \{i, j\} \in E$ existe um custo de deslocamento c_e e com cada

vértice $i \in V$ existe um prêmio p_i, não negativo, a ser coletado se o vértice for visitado e uma penalidade γ_i a ser paga se o vértice não for visitado. Assumindo que o vértice 0 seja o ponto de partida do caixeiro, este possui prêmio nulo e penalidade infinita.

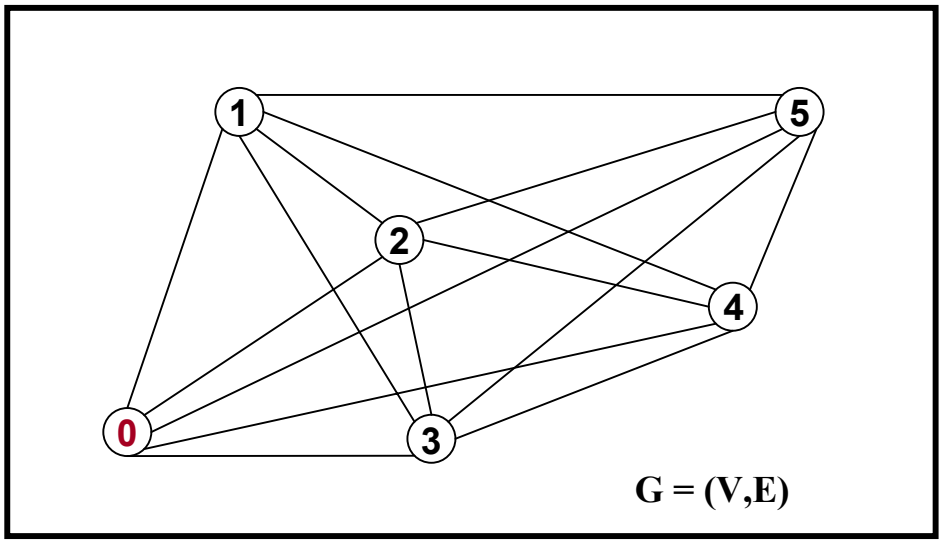

FIGURA 2.1 – Uma representação do PCVCP.

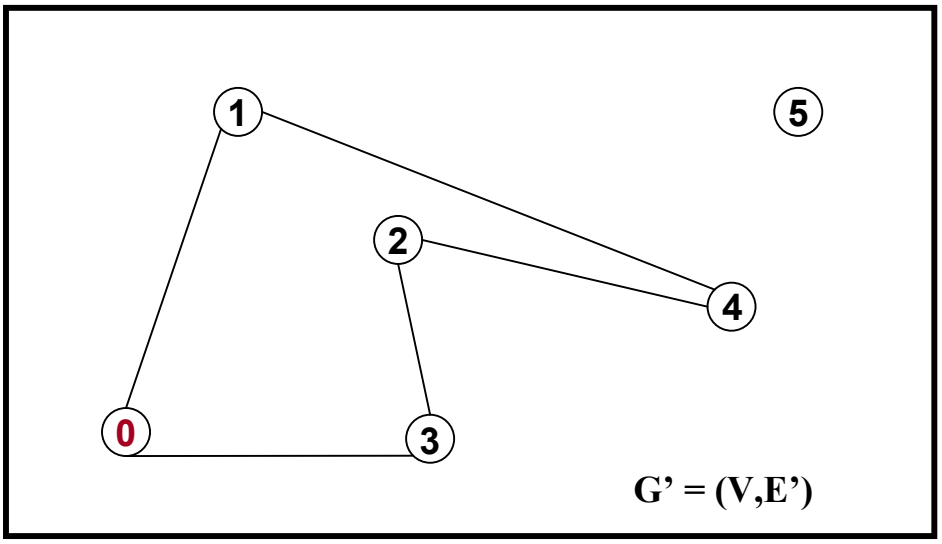

FIGURA 2.2 – Uma possível solução para o PCVCP.

A partir do grafo G = (V,E) apresentado na figura 2.1 uma possível solução para o PCVCP pode ser ilustrada na figura 2.2. Construindo um sub-grafo G' = (V,E') objetivando minimizar o somatório dos custos de deslocamento e o somatório das penalidades pagas, além de coletar uma quantidade de prêmios maior que o prêmio mínimo. A solução apresentada na figura 2.2 possui a seguinte sequência de visitas $S = \{0, 1, 4, 2, 3, 0\}$, onde o caixeiro tem um custo de deslocamento igual a $c_{01} + c_{14} +$

$c_{42} + c_{23} + c_{30}$, coletando um prêmio igual $p_0 + p_1 + p_4 + p_2 + p_3$, além de pagar uma penalidade igual a γ_5 por não ter visitado o vértice 5.

CAPÍTULO 3

REVISÃO

O PCVCP 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 *cutting planes* com uma relaxação de programação linear a ser resolvida através do método *simplex*. Balas e Martin (1985) desenvolveram um *software* para programação diária de uma fábrica de lâminas de aço, que utilizava a combinação de várias heurísticas para procurar boas soluções para o PCVCP.

Fischetti e Toth (1988) apresentam diferentes *lower bounds* para o PCVCP, obtidos através da exploração de diferentes relaxações do problema, utilizando relaxação Lagrangeana e relaxação baseada em disjunção. Entretanto, as soluções geradas pelas relaxações são geralmente inviáveis para o PCVCP, pois as rotas podem conter mais de uma sub-rota e/ou o total de prêmios coletados podem ser menor que o prêmio mínimo pré-estabelecido. Eles descrevem também um algoritmo *branch and bound* para obter a solução ótima para o PCVCP, sendo que este foi aplicado apenas à instâncias pequenas.

Bienstock et al. (1993) apresentaram o primeiro algoritmo aproximantivo para o PCVCP, consistia em um algoritmo híbrido combinando Programação Linear e heurística. Primeiramente resolve-se uma relaxação do PCVCP através de Programação Linear. Em seguida transforma-se a solução relaxada obtida em uma solução viável, utilizando uma heurística baseada no algoritmo de Christofides (Christofides, 1976). A performance deste algoritmo no pior caso fica a $2,5\%$ do ótimo.

Goemans e Williamson (1995) desenvolveram um algoritmo 2-aproximativo para uma versão do PCVCP, onde o objetivo era minimizar o custo da rota e a penalidade paga nos vértices não visitados, não levando em consideração o prêmio mínimo a ser coletado. Tendo uma performance melhor que a apresentada em (Bienstock et al., 1993).

Dell'Amico et al. (1995) apresentam uma nova abordagem para obter *lower bounds* para o PCVCP. 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.

Dell'Amico et al. (1998) propuseram um outro algoritmo para PCVCP, utilizando a relaxação Lagrangeana apresentada anteriormente, e um algoritmo baseado numa heurística clássica para o PCV, conhecida como heurística de inserção mais barata, para

transformar a solução relaxada encontrada em uma solução viável, através da inserção de vértices com melhores economias na rota até que o prêmio mínimo seja coletado. Propuseram ainda, utilizar uma heurística chamada de Extensão e Colapso para melhorar a solução viável obtida.

Lopez et al. (1998) apresentaram uma abordagem específica para um problema encontrado em indústrias de aço quando programam sua produção de lâminas de aço, chamado *The hot strip mill production scheduling problem*. Este problema pode ser modelado como uma generalização do PCVCP. Os autores apresentam uma formulação matemática para o problema e propõem-se um método heurístico baseado em Busca Tabu para encontrar boas soluções para o problema.

Gomes et al. (2000) apresentam uma metaheurística híbrida para solucionar o PCVCP, combinando a metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP) e a heurística *Variable Neighborhood Descent* (VND) para realizar a fase de busca local do GRASP. O VND realiza três procedimentos de busca local em vizinhanças diferentes, sendo eles: 3-opt, uma sequência de movimentos de remoção e inserção de vértices (*SeqDrop-SeqAdd*), e por fim realiza somente movimentos de remoção (*SeqDrop*). Estes movimentos, remoção e inserção de vértices, são realizados segundo a heurística propota em (Dell'Amico et al., 1998).

Melo (2001) e (Melo e Martinhon, 2004) introduzem o conceito de distribuição de soluções progressivas (GRASP-Progressivo) como fase de construção para metaheurísticas híbridas, que combinam GRASP e VNS, para solucionar aproximadamente o PCVCP. Com base nas aplicações de GRASP relatadas na literatura, observa-se que a utilização de um único valor fixo para o parâmetro α, que controla a construção de soluções gulosas, acarreta um processo exaustivo de calibragem, além de associar este às instâncias consideradas. Por isso, os autores propõe utilizar todas as variações para $\alpha \in [0, 1]$, de maneira que tenha-se uma distribuição progressiva de soluções na fase de construção, existindo no mínimo 1 solução para α próximo a 0 e no máximo n soluções para α próximo de 1. Na fase de busca local várias combinações de estruturas de vizinhanças diferentes são apresentadas para refinar a solução.

Torres e Brito (2003) apresentam uma nova formulação matemática para o PCVCP baseada na formulação apresentada em Balas (1989). Nesta formulação é proposto um novo conjunto de restrições para evitar a formação de sub-rotas na solução. Estas restrições foram desenvolvidas através da inclusão de variáveis de fluxo, utilizando para isso um número polinomial de restrições. Com esta forma de atender esta condição o tempo computacional para resolver este problema tende a ser menor, uma vez que

as formulações mais comumente usadas no PCV exigem um número exponencial de restrições para atendendimento desta condição.

Chaves et al. (2003, 2004a,b) desenvolveram trabalhos abordando duas modelagens. Uma modelagem matemática, dita exata, onde o PCVCP é resolvido de forma ótima para problemas de pequenas dimensões, e uma modelagem heurística, combinando as metaheurísticas GRASP e VNS, onde foi investigado o uso de filtragem na fase de construção, ou seja, a utilização de soluções elites. No VNS utilizou-se estruturas de vizinhança baseadas na troca de vértices, através de inserções e remoções aleatórias, e também através de trocas na sequência de visita dos vértices. Na fase de refinamento do vizinho gerado pelo VNS utilizou-se o método VND, sendo que este faz uso de heurísticas de refinamento baseadas em inserções e remoções de vértices (heurísticas *Add-step* e *Drop-step*) e troca de arestas (heurística 2-*opt*). Encontrando bons resultados, uma vez que, na comparação dos resultados encontrados nas modelagens exata e heurística, verifica-se que o algoritmo heurístico sempre encontraram o ótimo para as instâncias onde este é conhecido.

Existem na literatura problemas que são semelhantes ao PCVCP, mas que diferem deste em alguns detalhes. Alguns destes problemas são apresentados a seguir:

- *Resource Constrained Travelling Salesman Problem (RCTSP)* (Pekny e Miller, 1990): dado um conjunto de n cidades com um custo de deslocamento e um recurso consumido para ir de uma cidade para outra. O objetivo é encontrar uma rota com custo mínimo de deslocamento que não consuma mais que uma certa quantidade de recurso pré-definida.

- *Orieenting Problem (OP)* (Tsiligirides, 1984): dado um conjunto de vértices, onde existe associado a cada vértice um valor em prêmio que é coletado caso o vértice seja visitado, exceto os vértices origem e destino. O objetivo do OP é gerar uma rota iniciando na origem e finalizando no destino, visitando um subconjunto de vértices de modo que a soma dada pelo total de prêmios coletados seja maximizada e a rota seja completada dentro de um limite de tempo pré-estipulado.

- *Prize-Collecting Steiner Tree Problem (PCSTP)* (Canuto et al., 2001): dado um grafo conectado e não direcionado, com um peso associado às arestas e um prêmio não negativo associado aos vértices, o objetivo do PCSTP é encontrar uma sub-árvore que minimize a soma dos pesos das arestas mais os prêmios dos vértices não visitados. O PCSTP tem uma importante aplicação em telecomunicação.

- *Prize-Collecting Travelling Salesman Problem with Time Windows (TW-TSP)* (Bar-Yehuda et al., 2003): dado um conjunto de clientes com um tempo de deslocamento entre eles, cada cliente possui um intervalo de tempo no qual ele pode ser atendido e um prêmio que será coletado caso ele seja atendido. O objetivo do TW-TSP é encontrar um rota coletando o máximo de prêmio possível, sem violar a restrição de janela de tempo.

CAPÍTULO 4

MÉTODOS DE SOLUÇÃO

Muitos problemas práticos das mais diversas áreas, podem ser enquadrados como problemas de otimização combinatória, tais como roteamento e escalonamento de veículos, programação de horários para empregados, localização de facilidades (por exemplo escolas, hospitais, estação de bombeiros), problemas de corte e empacotamentos, entre outros.

Estes problemas são classificados na literatura como $NP-difíceis$, isto é, não são conhecidos algoritmos que os resolvam em tempo polinomial. As soluções ótimas para esses tipos de problemas poderiam ser encontradas através de uma enumeração completa de todas as soluções possíveis, porém, mesmo com o avanço tecnológico dos computadores nas últimas décadas, isto torna-se impratícavel à medida que o tamanho do problema aumenta.

Exitem formas de tornar o método de enumeração mais eficiente, utilizando por exemplo as técnicas *branch and bound* ou *branch and cut*, reduzindo o número de soluções a analisar no espaço de busca. Com isto, pode ser possível resolver problemas de dimensões um pouco mais elevadas. Entretanto, certamente haverá uma dimensão acima da qual o problema torna-se intratável computacionalmente.

Assim sendo, em problemas de otimização combinatória, o uso de métodos exatos se torna bastante restrito. Porém, na maioria dos problemas reais, é suficiente encontrar uma boa solução para o problema, ao invés do ótimo global, o qual somente pode ser encontrado após um considerável esforço computacional.

Por este motivo, problemas desta natureza são comumente abordados através de heusrísticas, que são métodos mais inteligentes para encontrar boas soluções a um custo computacional mais razoável.

Neste trabalho, serão utilizados vários métodos que já foram adotados para solucionar diversos problemas de otimização. Esses métodos são descritos a seguir.

4.1 Programação Linear

Programação Linear (PL) é uma técnica de otimização utilizada para encontrar o lucro máximo ou o custo mínimo em situações nas quais temos diversas alternativas de escolha sujeitas a algum tipo de restrição ou regulamentação (Prado, 1999).

Esta técnica se consolidou em 1947, quando George Dantzig desenvolveu o método *Simplex*, capaz de resolver qualquer tipo de problema de PL.

A PL procura representar problemas reais em modelos matemáticos que preservam uma equivalência adequada com a realidade. Os modelos matemáticos, ou formulações matemáticas, consistem em uma função objetivo, que deve ser minimizada ou maximizada, e de um conjunto finito de restrições. Tanto a função objetivo como as restrições são equações/inequações lineares ou de primeiro grau. Caso as variáveis do modelo admitam somente valores inteiros, a PL é dita Programação Linear Inteira (PLI).

Na prática a PL tem sido aplicada nas mais diversas áreas como dosagem (fabricação de rações, ligas metálicas, minérios), transporte, investimento financeiro, alocação de recursos (fábricas, máquinas), localização industrial, fluxo em redes, entre outros.

4.2 Heurísticas

As heurísticas ou algoritmos heurísticos foram desenvolvidos com o propósito de resolver problemas de elevado nível de complexidade em tempo computacional razoável. Ao se pensar em um problema combinatório, uma opção seria analisar todas as combinações possíveis para conhecer a melhor. Se o problema possui um universo de dados pequeno, realmente esta é a maneira correta de se buscar a melhor solução, mas os problemas reais, normalmente, possuem um número de combinações muito extenso, o que torna inviável a análise de todas as combinações, uma vez que o tempo computacional exigido fica impraticável. As heurísticas procuram encontrar soluções próximas da otimalidade em um tempo computacional razoável, sem, no entanto, conseguir definir se esta é a solução ótima, nem quão próxima ela está da solução ótima.

4.2.1 *Add-step*

Add-step (Gomes et al., 2000) é uma heurística míope utilizada para gerar soluções viáveis com uma certa qualidade de maneira eficiente (rápida), utilizando para tal, a regra de inserção proposta em Dell'Amico et al. (1998).

No procedimento *Add-step*, inicialmente, insere o vértice origem na rota R e tem-se R partindo e retornando à origem. Em seguida, escolhe-se um vértice $j \notin R$ para ser inserido em R, de forma que o custo da rota resultante seja mínimo, ou seja, inserir $j \notin R$ entre v_i e $v_{i+1} \in R$ com a função de economia definida a seguir.

$$j, v_i : \min \left\{ c_{v_i j} + c_{j v_{i+1}} - c_{v_i v_{i+1}} - \gamma_j : j \notin R, v_i \in R \right\} \tag{4.1}$$

A função 4.1 é composta pelos custos de deslocamento das arestas (v_i, j) e (j, v_{i+1}) que serão percorridas caso o vértice j seja inserido entre v_i e v_{i+1}, menos o custo de deslocamento da aresta (v_i, v_{i+1}) que deixará de ser percorrida e a penalidade do vértice j que deixará de ser paga, uma vez que o vértice será visitado.

Note que, caso o valor da função de economia seja negativo, obtêm-se uma melhora (diminuição) no valor da função objetivo do problema.

O método termina quando o somatório dos prêmios coletados nos vértices pertencentes a R for igual ou maior que o prêmio mínimo, p_{min}, e não haja mais vértice $j \notin R$ com economia negativa.

4.2.2 *Drop-step*

Drop-step (Gomes et al., 2000) também é uma heurística míope que parte de uma solução viável, e utiliza uma abordagem oposta à do algoritmo de inserção, ou seja, a cada iteração retira-se um vértice segundo o cálculo de uma função gulosa.

Definida uma solução inicial, com uma rota R que contenha todos ou alguns vértices, escolhe-se um vértice $j \in R$, que forneça o custo mínimo de rota resultante, removendo j da rota. A economia de remoção do vértice j é definida pela função 4.2.

$$j : \min \{c_{v_a v_s} + \gamma_j - c_{v_a j} - c_{j v_s} : j, v_a, v_s \in R\} \tag{4.2}$$

A função 4.2 é composta pelo custo de deslocamento entre os vértices v_a e v_s, que representam, respectivamente, o antecessor e o sucessor do vértice j na rota R, e pela penalidade do vértice j que deixará de ser visitado, menos os custos das arestas (v_a, j) e (j, v_s) que não serão mais percorridas.

Novamente, observa-se que, quando a economia é negativa a remoção do vértice j proporcionará uma redução no valor da função objetivo.

O procedimento é executado enquanto existir algum vértice com economia negativa e a retirada desse vértice não acarretar uma solução inviável, ou seja, o somatório dos prêmios dos vértices pertencentes à rota for menor que o prêmio mínimo pré-estabelecido.

4.2.3 2-Opt

Partindo de uma solução viável, tem-se uma rota com alguns (ou todos) vértices, e sabe-se qual a seqüência em que estes vértices são visitados. O desejo agora é conseguir

uma possível melhora no valor da função objetivo através da tentativa de mudança na ordem de visitas, o que pode ser conseguido aplicando a heurística 2-*Optimal* ou 2-Opt.

A heurístia 2-Opt foi proposta primeiramente por Croes (1958) sendo baseada em trocas entre pares de arestas de grafos que representem soluções para problemas de permutação. O movimento de troca remove duas arestas, quebrando o circuito em dois caminhos, e os reconecta da outra maneira possível. São verificadas todas as possíveis trocas para o grafo, realizando aquela que fornecer o melhor ganho.

Movimentos 2-Opt somente são realizados se as novas arestas incluídas possuírem custo menor que as removidas. É esperado que a substituição de arestas de maior custo por outras de menor custo reduza o custo total do circuito ou seja a distância total entre as cidades percorridas pelo caixeiro viajante.

Nas heurísticas do tipo k-Opt (2-Opt, 3-Opt, etc) à medida que o valor de k aumenta, em geral, aumenta também a probabilidade de se alcançar a solução ótima. Entretanto, o custo computacional também cresce rapidamente com o valor de k, pois a complexidade deste algoritmo é $O(n^k)$. Neste trabalho optou-se por utilizar a heurística 2-Opt, por esta ser eficiente e computacionalmente mais barata.

A heurística 2-Troca também se oferece como uma opção para procurar soluções melhores através da mudanças na ordem de visitas, com complexidade similar a 2-Opt ($O(n^2)$). A heurística 2-Troca se difenrencia da heurística 2-Opt por tentar realizar trocas entre os vértices que pertencem à rota, e não entre as arestas. A figura 4.1 mostra as heurísticas 2-Opt e 2-Troca aplicadas à uma mesma solução inicial.

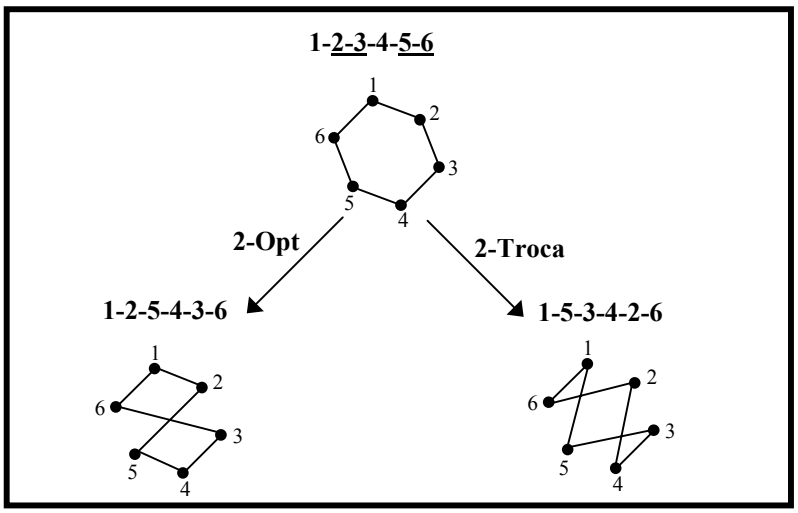

FIGURA 4.1 – Exemplo de movimentos 2-Opt e 2-Troca.
FONTE: Oliveira (2004)

4.2.4 *Path-relinking*

Path-relinking foi originalmente proposto por Glover (1996) como uma estrátegia de intensificação usada para explorar trajetórias conectando solução elites obtidas através de busca tabu ou busca *scatter*.

Partindo de uma ou mais soluções, caminhos no espaço de solução conduzindo para outras soluções são gerados e explorados na busca por soluções melhores. Para gerar estes caminhos, movimentos são selecionados para introduzir atributos na solução corrente, s_0, que estão presentes na solução guia, s_g, que geralmente tem alta qualidade. *Path-relinking* pode ser visto como uma estratégia que busca incorporar atributos de soluções que tenham alta qualidade, por favorecer estes atributos nos movimentos de seleção.

O procedimento *path-relinking* começa calculando a diferença simétrica entre as duas soluções, ou seja, o conjunto de movimentos necessáios para se chegar a s_g (solução guia) de s_0 (solução corrente). Um caminho de soluções é gerado ligando s_0 e s_g. A melhor solução, s^*, no caminnho é retornada pelo procedimento. A cada passo, é examinado todos os movimentos da solução corrente s_0 que ainda não foram selecionados e escolhe-se aquele que resultar na solução de menor custo. O melhor movimento é realizado produzindo uma nova solução corrente e o conjunto de movimentos necessários é atualizado. O procedimento termina quando s_g é alcançado, ou seja, não exite mais diferença entre s_0 e s_g. A figura 4.2 ilustra o funcionamento do *path-relinking*.

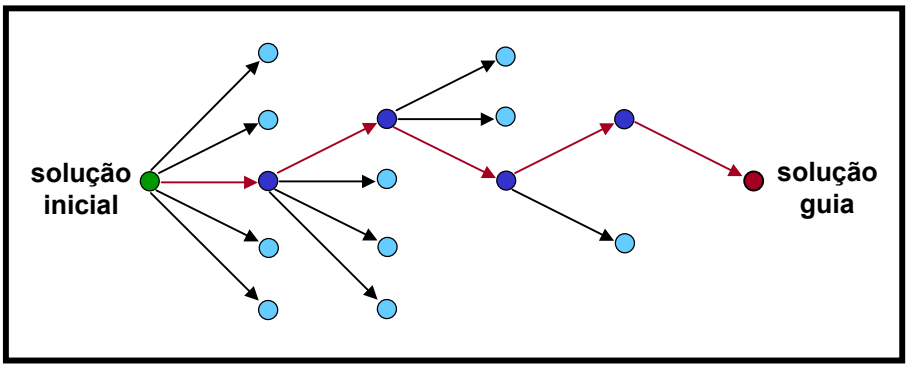

FIGURA 4.2 – Exemplo de movimentos do *path-relinking*

4.3 Metaheurísticas

Muitos trabalhos foram desenvolvidos nas últimas décadas com o sentido de melhorar os métodos heurísticos, sem no entanto prejudicar a sua principal característica, que é a flexibilidade. Estes trabalhos deram origem as estratégias comumente conhecidas como metaheurísticas.

Metaheurísticas são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima, consistindo na aplicação, em cada passo, de uma heurística subordinada, a qual tem que ser modelada para cada problema específico. A principal característica das metaheurísticas é a capacidade que estas possuem de escapar de ótimos locais.

4.3.1 GRASP

GRASP (*Greedy Randomized Adaptive Search Procedure*) (Feo e Resende, 1995) é uma metaheurística para encontrar soluções aproximadas para problemas de otimização combinatória. Desde que foi proposta, GRASP vem se desenvolvendo continuamente, e tem sido aplicada a uma gama extensiva de problemas da área.

GRASP é um processo iterativo, no qual cada iteração consiste em duas fases distintas: a fase de construção, onde uma solução viável é construída, e a fase de busca local, onde um ótimo local na vizinhança da solução inicial construída é pesquisado. A melhor solução encontrada ao longo de todas as iterações GRASP realizadas é retornada como resultado. A figura 4.3 apresenta um pseudo-código para o algoritmo GRASP.

```
procedimento GRASP( )
1   para (Critério de parada GRASP não satisfeito) faça
2        Construção(Solução);
3        BuscaLocal(Solução);
4        AtualizarSolução(Solução, MelhorSoluçãoEncontrada);
5   fim-para
6   retorne(MelhorSoluçãoEncontrada)
fim-GRASP
```

FIGURA 4.3 – Pseudo-código do GRASP.
FONTE: (Feo e Resende, 1995)

Na fase de construção, uma solução é iterativamente construída, elemento por elemento. A cada iteração dessa fase, os próximos elementos candidatos a serem incluídos na solução são colocados em uma lista de candidatos (C), seguindo um critério de ordenação pré-determinado. Esse processo de seleção é baseado em uma função adaptativa gulosa $h : C \mapsto \Re$, que estima o benefício da seleção de cada um dos elementos. A heurística é adaptativa porque os benefícios associados com a escolha de cada elemento são atualizados em cada iteração da fase de construção para refletir as mudanças promovidas pela seleção do elemento anterior.

O componente probabilístico do procedimento reside no fato de que cada elemento

é selecionado de forma aleatória a partir de um subconjunto restrito formado pelos melhores elementos que compõem a lista de candidatos. Este subconjunto recebe o nome de lista de candidatos restrita (LCR). Esta técnica de escolha permite que diferentes soluções sejam geradas em cada iteração GRASP. Um parâmetro $\alpha \in [0, 1]$ controla o quão gulosa será uma solução construída na fase de construção. Um valor α igual a 0 faz gerar soluções puramente gulosas, enquanto que α igual a 1 faz produzir soluções totalmente aleatórias. A figura 4.4 descreve a fase de construção GRASP.

```
procedimento Construção( Solução )
1   Solução = { };
2   para (Solução não construída) faça
3        ProduzirLCR(LCR);
4        s = SelecionarElementoAleatoriamente(LCR);
5        Solução = Solução ∪ {s};
6        AtualizarConjuntoCandidatos( );
7   fim-para
fim-Construção
```

FIGURA 4.4 – Pseudo-código da Fase de Construção.
FONTE: (Feo e Resende, 1995)

Assim como em muitas técnicas determinísticas, as soluções geradas pela fase de construção GRASP provavelmente não são localmente ótimas com respeito à definição de vizinhança adotada. Daí a importância da fase de busca local, a qual objetiva melhorar a solução construída.

A eficiência da busca local depende da qualidade da solução construída. A fase de construção tem então um papel importante na busca local, uma vez que as soluções construídas constituem bons pontos de partida para a busca local, permitindo assim acelerá-la.

Normalmente as aplicações de GRASP relatadas na literatura, utilizam sempre um único valor fixo para o parâmetro α durante a execução de todas as iterações do procedimento. Este valor fixo é simplesmente arbitrado ou resultante de um processo de calibragem para as instâncias do objeto de estudo, em geral próximo do valor correspondente a escolha puramente gulosa. O processo de calibragem, que além de dispendioso e exaustivo, esta associado as instâncias consideradas.

Procedimentos GRASP mais sofisticados incluem estratégias adaptativas para o parâmetro α. Um destes procedimentos é conhecido como GRASP Reativo (Prais e Ribeiro, 2000) no qual o parâmetro α é auto-ajustado à medida que iterações vão

sendo realizadas, em função da qualidade das soluções obtidas nas iterações anteriores. Esta estrátegia produz soluções melhores do que aquelas obtidas considerando α fixo, entretanto, levam em geral a tempos de processamento maiores, já que soluções de qualidade inferior são construídas, o que torna a busca local mais lenta.

4.3.2 VNS

O método de Pesquisa em Vizinhança Variável (*Variable Neighborhood Search*, VNS) inicialmente proposto por Mladenovic e Hansen (1997), combina busca local com mudanças sistemáticas de vizinhança, procurando escapar de ótimos locais. Ao contrário da maioria dos outros métodos de busca local, o VNS não segue uma trajetória, mas explora vizinhanças cada vez mais distantes da solução corrente, pelo fato de gerar vizinhos aleatoriamente, movendo-se para a nova solução se e somente se uma melhora for produzida. Além disso, um método de busca local é aplicado repetidamente para transformar a solução vizinha em um ótimo local. Um pseudo-algoritmo do VNS é mostrado na figura 4.5.

```
procedimento VNS ( )
1   k_max = número de vizinhanças;
2   solução inicial s;
3   enquanto (critério de parada não satisfeito)
4     k ← 1;
5     enquanto (k ≤ k_max)
6       gerar um vizinho s' qualquer na vizinhança N^k(s);
7       Aplicar um método de busca local em s' obtendo
8       um ótimo local (s");
9       se (s" for melhor que s) faça
10        s ← s";
11        k ← 1;
12      senão
13        k ← k + 1;
14    fim-enquanto
15  fim-enquanto
16  retornar s;
fim-VNS
```

FIGURA 4.5 – Pseudo-código do VNS.
FONTE: adaptado (Chaves et al., 2003)

Considerando um conjunto finito pré-definido de estruturas de vizinhança $N^k, (k = \{1, \ldots, k_{max}\}$, e com $N^k(s)$ sendo o conjunto de soluções da $k-$ésima vizinhança da

solução corrente s. O método inicializa com uma solução inicial qualquer e a cada iteração é gerado aleatoriamente um vizinho, s', dentro da vizinhança N^k da solução corrente ($s' \in N^k(s)$). Aplica-se então um método de busca local no vizinho gerado, encontrando uma solução que representa um ótimo local, s''. Se este ótimo local for melhor que a solução corrente ($s'' < s$) a busca continua a partir deste, recomeçando da primeira estrutura de vizinhança. Caso contrário, continua-se a busca a partir da próxima estrutura de vizinhança, N^{k+1}. Esta heurística é encerrada quando uma condição de parada for atingida, tal como o tempo máximo de processamento ou o número máximo de iterações consecutivas sem melhora da solução corrente.

4.3.3 Algoritmos Genéticos

Os primeiros estudos sobre Algoritmos Genéticos (AGs) foram desenvolvidos por Holland (1975) e Goldberg (1989). Trata-se de uma metaheurística que se fundamenta em uma analogia com processos naturais de evolução, nos quais, dada uma população, os indivíduos com melhores características genéticas têm maiores chances de sobrevivência e de produzirem filhos cada vez mais aptos, enquanto indivíduos menos aptos tendem a desaparecer.

Nos AGs, cada cromossomo (indivíduo da população) está associado a uma solução do problema e cada gene está associado a um componente da solução. Um mecanismo de reprodução, baseado em processos evolutivos, é aplicado sobre a população com o objetivo de explorar o espaço de busca e encontrar melhores soluções para o problema.

Para melhorar o desempenho dos AGs na resolução de problemas reais, algoritmos de busca que levam em consideração as propriedades do problema têm sido acoplados nos AGs. A grande maioria dos trabalhos recentes conservam as características de evolução natural, mas utilizam uma grande variedade de heurísticas específicas para o problema em questão.

4.3.4 Algoritmo de Treinamento Populacional

O Algoritmo de Treinamento Populacional (ATP) é uma técnica evolutiva proposta por Oliveira (2004) e Oliveira e Lorena (2005), que surgiu de uma derivação de uma outra técnica evolutiva chamada Algoritmo Genético Construtivo (AGC), Lorena e Lopes (1996).

O termo treinamento indica um processo no qual um modelo computacional, uma vez estimulado, consegue assimilar ou se adaptar a um determinado ambiente inicialmente desconhecido para ele, através do ajuste dos parâmetros livres segundo alguma medida

de desempenho ou comportamento esperado (Oliveira, 2004).

O ATP consiste em induzir uma população de indivíduos a assumir uma determinada característica, tornando-a adaptada a esta característica. A indução é realizada privilegiando os indivíduos mais adaptados e penalizando os menos. Heurísticas específicas sobre o problema tratado são utilizadas para determinar as características desejadas no treinamento. Estas heurísticas são utilizadas para buscar novas soluções viáveis melhores que as originais. Se houver uma solução heurísticamente melhor, diz-se que o indivíduo original não está bem-adaptado à heurística de treinamento empregada. Caso contrário, o indivíduo é o melhor dentro da vizinhança estabelecida pela heurística, devendo participar o máximo de gerações possíveis do processo de evolução. A figura 4.6 apresenta o ATP.

```
procedimento ATP ( )
1   GerarPopulaçãoInicial (P);
2   enquanto (critério de parada não satisfeito)
3         selecionar S_base e S_guia
4         S_k = recombinar(S_base e S_guia)
5         se (probabilidade de mutação aceita)
6             mutação(S_k);
7         fim-se
8         avaliar (S_k);
9         avaliar (vizinhança(S_k));
10        calcular (δ(S_k));
11        se (δ(S_k) > τ )
12            atualizar (população(P,S_k));
13        fim-se
14        incrementar (τ);
15        enquanto (existir δ(S) ≤ τ)
16            eliminar (S);
17        fim-enquanto
18  fim-enquanto
fim-ATP
```

FIGURA 4.6 – Pseudo-código do ATP.
FONTE: adaptada (Oliveira, 2004)

O ATP trabalha com uma população dinâmica de indivíduos, sendo que, inicialmente a população é gerada aleatoriamente. Durante o processo evolutivo, a cada geração um novo indivíduo é gerado através do cruzamento entre dois indivíduos já existentes na população, privilegiando os indivíduos mais bem adaptados. O indivíduo gerado

eventualmente pode sofrer alguma mutação, modificando suas características herdadas.

Cada novo indivíduo é avaliado por duas funções, f e g, a primeira avalia a qualidade do indivíduo e a segunda aplica a heurística de treinamento para avaliar a vizinhança do indivíduo, sendo a melhor solução encontrada atribuída como valor de g.

A adaptação de um indivíduo pode ser medida através do $ranking(\delta)$, equação 4.3, que é composta, além do componente referente a adaptação do indivíduo em relação à heurística de treinamento $(f-g)$, por um componente que fornece a distância, em termos de avaliação, entre o indivíduo s_k e a constante G_{max}. Sendo G_{max} uma estimativa de um limite superior para todos os possíveis valores que as funções f e g podem assumir, e d uma constante que tem o papel de equilibrar os componentes da equação. Em problemas de minimização, os indivíduos melhores adaptados são aqueles com baixos valores da heurística de treinamento e que estejam bem-adaptados à esta.

$$\delta\left(s_k\right) = d.\left[G_{\max} - g\left(s_k\right)\right] - \left|f\left(s_k\right) - g\left(s_k\right)\right| \tag{4.3}$$

A população é controlada dinamicamente por um limiar de rejeição, τ, que durante o processo evolutivo é atualizado através de incrementos adaptativos que consideram o tamanho atual da população, $|P|$, bem como a atual faixa de valores de $ranking$ (do melhor, δ_1, ao pior, $\delta_{|P|}$), além do número de gerações que faltam para terminar o processo de evolução, RG. O limiar de rejeição é dado por:

$$\tau_{i+1} = \tau_i + \xi.\left|P\right|.\frac{\left(\delta_1 - \delta_{|P|}\right)}{RG} \tag{4.4}$$

onde ξ é uma constante que controla a velocidade do processo de esvaziamento da população, quanto menor mais lento é o processo evolutivo.

No fim de cada geração os indivíduos menos adaptados $(\delta(s_k) \leq \tau)$ são eliminados da população. No ATP a população tende a crescer, inicialmente, aceitando todos os novos indivíduos. Com o passar do tempo, o aumento do valor de τ determina uma quantidade cada vez maior de indivíduos a serem eliminados.

4.3.5 *Evolutinary Clustering Search* (ECS)

Trata-se de uma técnica proposta por Oliveira (2004) e Oliveira e Lorena (2004), sendo derivada de Algoritmos Evolutivos. O *Evolutionary Clustering Search* (ECS), ou busca

evolutiva através de agrupamentos, consiste num processo de agrupamento de indivíduos para guiar uma busca evolutiva, sendo usado para gerar um conjunto de soluções de referência para regiões supostamente promissoras.

A idéia de detectar regiões promissoras se justifica pelo fato de possibilitar a aplicação de busca local somente em indivíduos realmente promissores, diminuindo assim o número de chamadas à função objetivo que, para problemas reais podem ter um custo significativo.

No ECS um processo de agrupamento iterativo trabalha simultaneamente com um algoritmo evolutivo, identificando grupos de indivíduos que merecem especial interesse. As regiões destes grupos de indivíduos 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 da busca local.

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

O centro é um indivíduo (ou solução) representante do *cluster*, que identifica a sua localização dentro do espaço de busca. O raio estabelece a distância máxima, a partir do centro, até a qual um indivíduo pode ser associado ao *cluster*.

Em um problema de otimização combinatória, o raio pode ser definido em termos de número de movimentos necessários para transformar uma solução candidata em outra dentro de uma vizinhança definida por uma heurística qualquer.

A estratégia β é uma sistemática de intensificação de busca, na qual indivíduos de um *cluster* interagem entre si, ao longo do processo de agrupamento, gerando novos indivíduos na mesma região.

O ECS consiste em quatro componentes com atribuições diferentes e conceitualmente independentes. São eles:

- um algoritmo evolutivo (AE);
- um agrupador iterativo (AI);
- um analisador de agrupamentos (AA);
- um algoritmo de otimização local (AO);

A figura 4.7 ilustra os quatro componentes, a população de indivíduos e os *clusters* que interagem entre si.

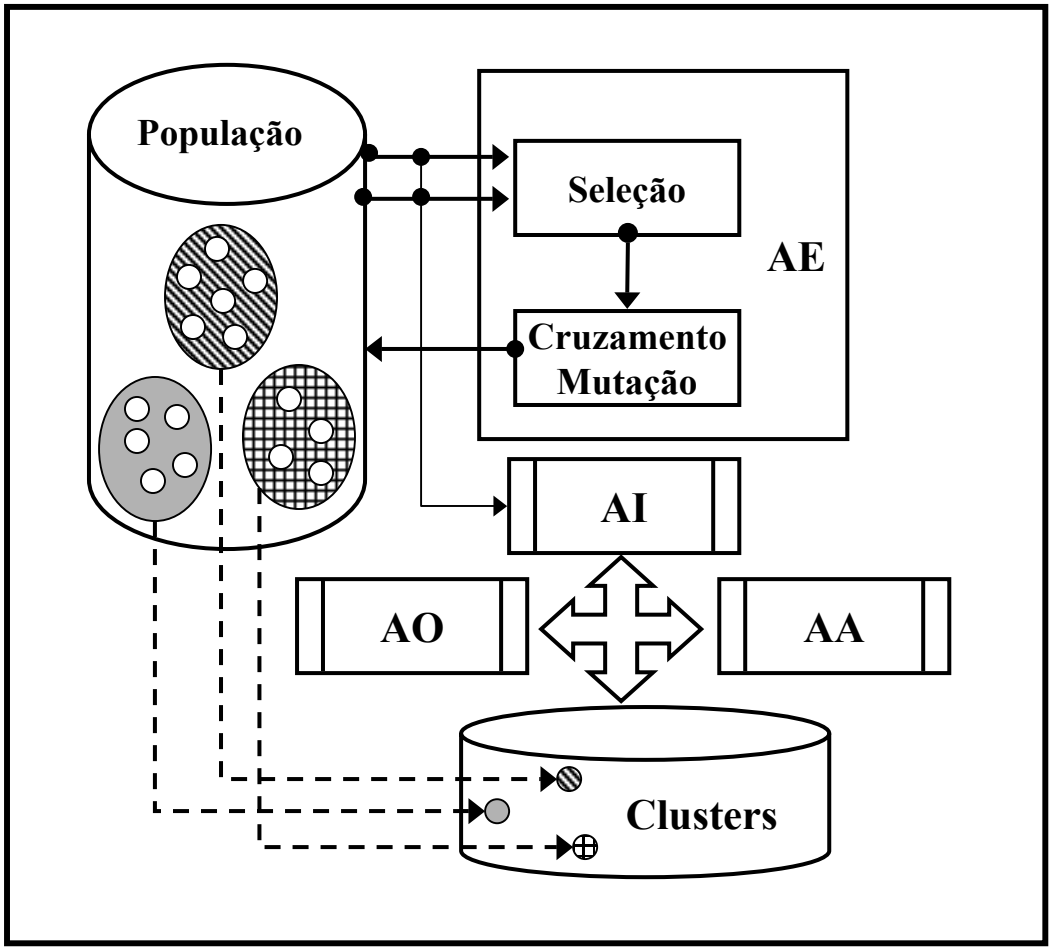

FIGURA 4.7 – Diagrama Conceitual do ECS.
FONTE: (Oliveira, 2004)

O algoritmo evolutivo (AE) trabalha como um gerador de soluções de tempo integral. A população evolui independentemente dos componentes restantes. Indivíduos são selecionados, recombinados, e são atualizados para as próximas gerações. Simultaneamente, *clusters* são mantidos para representar estes indivíduos.

O agrupador iterativo (AI) é o núcleo do ECS, trabalhando como um classificador de informação que mantém no sistema apenas aquela que for relevante para o processo de intensificação de busca. Usa-se o termo informação pois os indivíduos não se agrupam diretamente, e sim a informação semelhante que eles representam. Toda informação selecionada por AE é lida por AI que tenta agrupá-la como uma informação conhecida. Se a informação for considerada suficientemente nova, ela é mantida como um centro em um novo *cluster*. Caso contrário, a informação é considerada redundante, causando perturbação no centro do *cluster* mais similar. Tal perturbação é chamada de assimilação e consiste basicamente em atualizar o centro com a nova informação recebida.

O analisador de agrupamentos (AA) provê uma análise de cada *cluster*, em intervalos regulares de gerações, indicando um provável grupo promissor. Tipicamente, a densidade do *cluster* é usada nesta análise, ou seja, o número de seleções ou atualizações que aconteceram recentemente no *cluster*. Um *cluster* com densidade alta deve possuir um centro promissor. O AA também é responsável pela eliminação de *clusters* com baixas densidades.

Uma forma de detectar se um *cluster* esta representando adequadamente uma região promissora, é verificar o número de vezes que este foi ativado durante uma dada geração t. Sendo que, se a população e os *clusters* estiverem uniformemente distribuídos, cada *cluster* deveria ser ativado NI vezes a cada geração. NI pode ser assim definido:

$$NI = \frac{NS}{|C_t|} \tag{4.5}$$

onde NS é o número de indivíduos selecionados ou atualizados durante o processo de reprodução e $|C_t|$ é número total de *clusters* na geração t.

Por fim, o algoritmo de otimização local (AO) é um módulo de pesquisa interno que provê a exploração de uma suposta região promissora. Este processo acontece depois que o componente AA tenha descoberto um *cluster* alvo. A busca local é realizada no indivíduo que esta no centro do *cluster*.

Este método foi proposto para a resolução de problemas de otimização numérica e alguns problemas de otimização combinatória, como problemas de seqüenciamento de padrões e problemas de satisfabilidade, sendo que as experiências computacionais demonstraram a boa competitividade do mesmo.

CAPÍTULO 5

PROPOSTA

Neste trabalho são propostas três abordagens para resolver o PCVCP. A primeira será através de Programação Linear (PL) visando encontrar a melhor solução para o problema. A segunda e a terceira serão baseadas em algoritmos híbridos, utilizando algoritmos de busca através de agrupamentos para a obtenção de boas soluções, eventualmente a ótima. Na segunda abordagem será implementado o *Evolutionary Clustering Search* (ECS), sendo este um algoritmo evolutivo híbrido. Na terceira abordagem será implementado uma adaptação do ECS, chamada *CS, que utilizará as metahurísticas *Greedy Randomized Adaptive Search Procedure* (GRASP) e *Variable Neighborhood Search* (VNS) ao invés do algoritmo evolutivo para gerar soluções para o processo de agrupamento. Estas abordagens serão descritas detalhadamente a seguir.

5.1 Programação Linear

A Programação Linear (PL) representa problemas reais em modelos matemáticos. O modelo matemático, ou formulação matemática, consiste de uma função objetivo, que deve ser minimizada ou maximizada, e de um conjunto finito de restrições, sendo ambas formadas por um número finito de variáveis de decisão lineares.

O PCVCP pode ser representado em um grafo completo não direcionado $G = (V, E)$, onde existe associado a cada vértice $k \in V$ um prêmio p_k e uma penalidade γ_k, e cada aresta $(i, j) \in E$ possui um custo de deslocamento. Sendo o vértice 0, sem perda de generalidade, assumido como vértice origem. Ressaltando que o vértice origem possui prêmio nulo ($p_0 = 0$) e penalidade infinita ($\gamma_0 = \infty$).

A formulação matemática apresentada neste trabalho é baseada nas formulações propostas por Balas (1989) e Torres e Brito (2003). Utiliza-se o conjunto de restrições propostas em Torres e Brito (2003) para evitar a formação de sub-rotas. A vantagem de utilizar esse conjunto de restrições é que o número de restrições utilizadas é polinomial, ao contrário das restrições comumente usadas no PCV que exigem um número exponencial de restrições para o atendimento desta condição.

Considere x_{ij} $(i, j \in V, i \neq j)$ sendo uma variável binária igual a 1 se a aresta (i, j) pertencer a solução, e x_{ij} igual a 0 caso contrário. A variável x_{ii}, $i \in V$, controla se o vértice i está presente na rota, assumindo valor 0 caso seja visitado e valor 1 caso contrário. A variável f_{ij} $(i, j \in V, i \neq j)$ é a quantidade de fluxo (valor em prêmios) escoada na aresta (i, j), sendo utilizada para evitar que a solução contenha sub-rotas.

A formulação matemática é apresentada a seguir, onde: $b_{ij} = \begin{cases} c_{ij}, \text{ se } i \neq j \\ \gamma_i, \text{ caso contrário} \end{cases}$

$$\min \sum_{i \in V} \sum_{j \in V} b_{ij} x_{ij} \tag{5.1}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V \tag{5.2}$$

$$\sum_{j \in V} x_{ji} = 1 \qquad \forall i \in V \tag{5.3}$$

$$\sum_{i \in V} p_i x_{ii} \leq \left(\sum_{i \in V} p_i \right) - p_{\min} \tag{5.4}$$

$$\sum_{j \in V \setminus \{0\}} f_{0j} = 0 \tag{5.5}$$

$$\sum_{j \in V \setminus \{i\}} f_{ij} = \sum_{j \in V \setminus \{i\}} f_{ji} + p_i x_{ii} \qquad \forall i \in V \setminus \{0\} \tag{5.6}$$

$$\sum_{j \in V \setminus \{0\}} f_{j0} = \sum_{j \in V \setminus \{0\}} p_j x_{jj} \tag{5.7}$$

$$f_{ij} > x_{ij} - 1 \qquad \forall i \in V \setminus \{0\}, \;\; \forall j \in V, \;\; i \neq j \tag{5.8}$$

$$\left(\sum_{j \in V} p_j \right) x_{ij} \geq f_{ij} \qquad \forall i \in V \setminus \{0\}, \;\; \forall j \in V \tag{5.9}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{5.10}$$

$$f_{ij} \geq 0 \qquad \forall i, j \in V \tag{5.11}$$

A função objetivo 5.1 procura minimizar o somatório dos custos de deslocamento e as penalidades pagas. As restrições 5.2 e 5.3 garantem que se vértice i for visitado ($x_{ii} = 0$) somente uma aresta pode sair e somente uma aresta pode chegar neste vértice, caso o vértice não seja visitado ($x_{ii} = 1$) nenhuma aresta pode chegar ou sair deste. A restrição 5.4 assegura que a soma dos prêmios coletados será maior que o prêmio mínimo, p_{min}. As restrições 5.5, 5.6 e 5.7 garantem a conectividade da solução, ou seja, evitam rotas desconexas da origem. As variáveis de fluxo f_{ij} impedem que sub-rotas sejam

criadas, associando a quantidade de prêmio do vértice visitado à aresta que sai deste. A quantidade de fluxo que sai do vértice origem tem que ser igual a 0, e a quantidade de fluxo que volta para o vértice origem tem que ser igual à soma dos prêmios coletados na rota. As restrições 5.8 e 5.9 conectam as variáveis x_{ij} e f_{ij}, fazendo com que a rota gerada por ambas seja a mesma. Sendo que a restrição 5.8 garante que se uma aresta fizer parte da solução, a quantidade de fluxo escoada por esta tem que ser maior que 0, e a restrição 5.9 assegura que o valor do fluxo escoado por uma aresta não será maior que o total de prêmios de todos os vértices. As restrições 5.10 asseguram a bivalência das variáveis x_{ij}. Por fim, as restrições 5.11 garantem que as variáveis f_{ij} sejam não-negativas.

Para resolução desta formulação, utilizou-se o *software* CPLEX versão 7.5 (ILOG, 2001) buscando encontrar a solução ótima para o PCVCP. Entretanto, pela natureza combinatorial do PCVCP, tal modelagem só consegue resolver instâncias pequenas do problema. Através de testes, verifica-se que esta modelagem já se torna inviável computacionalmente para instâncias com 50 vértices.

5.2 ECS aplicado ao PCVCP

Um dos maiores desafios dos métodos de otimização é definir estratégias eficientes para cobrir todo o espaço de busca, possibilitando a aplicação de busca local somente em regiões realmente promissoras. No *Evolutionary Clustering Search* (ECS) ou busca evolutiva através de agrupamentos, o objetivo é detectar regiões promissoras através de agrupamento de genótipos. Explorando estas regiões, através de heurísticas de busca local específicas, tão logo elas sejam encontradas.

Para resolução do PCVCP, o ECS foi implementado da seguinte forma:

- algoritmo evolutivo: Algoritmo de Treinamento Populacional (ATP).
- métrica de distância: heurística 2-Troca.
- processo de assimilação: *path-relinking*.
- busca local: 2-Opt.

A representação de um indivíduo será realizada 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 5.1 ilustra a representação de um indivíduo, onde a sequência de visitas é {1,3,0,4} e os vértices 2 e 5 não foram visitados.

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

FIGURA 5.1 – Representação de um indivíduo.

5.2.1 Componente AE

O componente AE, responsável por gerar soluções para abastecer os agrupamentos, será o Algoritmo de Treinamento Populacional (ATP) proposto por Oliveira (2004).

O ATP é iniciado com uma população de tamanho $|P_0|$ definida aleatoriamente. Todos os indivíduos são avaliados pelas funções f e g, sendo ordenados na população segundo o *ranking* δ, de onde são selecionados, recombinados e eventualmente sofrem mutação.

Através dos mecanismos evolutivos, a cada geração um número constante (NS) de indivíduos são selecionados. A seleção de indivíduos para recombinação é realizada através do algoritmo de seleção *base-guia* (Lorena e Furtado, 2001). A seleção é realizada privilegiando os indivíduos com maior *ranking* δ, selecionando dois indivíduos para recombinação. O primeiro indivíduo (s_{base}) é selecionado aleatoriamente entre os melhores indivíduos da população, e o segundo indivíduo (s_{guia}) é selecionado aleatoriamente dentro da população inteira.

A recombinação é fundamental para o bom desempenho do ATP. O operador de recombinação implementado foi baseado no cruzamento *Block Order Crossover (BOX)* (Syswerda, 1989). O indivíduo s_{base} e o indivíduo s_{guia} são combinados, através da cópia aleatória de blocos de ambos indivíduos, o que resulta na geração de um único filho s_k. Blocos copiados de um indivíduo não são copiados do outro, mantendo a viabilidade do indivíduo resultante. A figura 5.2 mostra um exemplo do *BOX*.

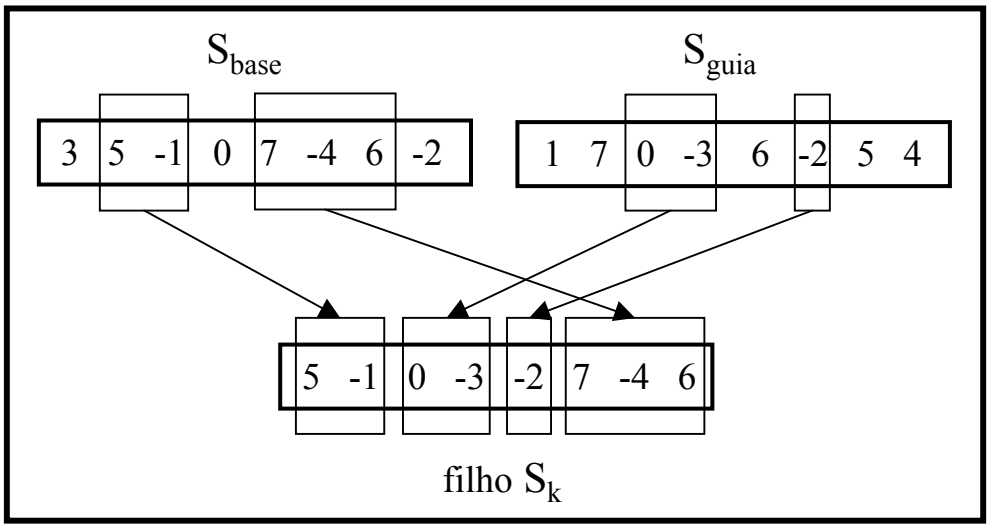

FIGURA 5.2 – Exemplo da recombinação *BOX*.
FONTE: adaptada (Oliveira, 2004)

Como operador de mutação implementa-se a heurística 2-Opt (seção 4.2.3), realizando

a troca das arestas que resultar no melhor valor de função objetivo, alterando assim, dois genes do cromossomo (ou seja, dois vértices da solução). Este operador introduz modificações no indivíduo s_k, permitindo que este apresente características diferentes das de seus pais, sendo executado com uma certa probabilidade ($\rho(m)$) durante o processo de reprodução.

A heurística de treinamento é usada para gerar um conjunto de soluções na vizinhança da solução s_k, que são avaliadas e a melhor é retornada como sendo o melhor vizinho de s_k, sendo o valor atribuido a função g. Neste trabalho, propõe-se utilizar o método *SeqAdd-SeqDrop* (Melo, 2001) como heurística de treinamento.

O método *SeqAdd-SeqDrop* baseia-se em técnicas de inserção de vértices (*Add-step*, seção 4.2.1) e remoção de vértices (*Drop-step*, seção 4.2.2). Funcionando de maneira iterativa, ou seja, através de sucessivas tentativas de troca da solução corrente por uma solução melhor na sua vizinhança. O que pode ser conseguido através de inserções de vértices com economia de inserção negativa, seguida de remoções de vértices com economia de remoção negativa, não permitindo que o prêmio mínimo deixe de ser coletado.

Cada indivíduo s_k gerado é avaliado pelas funções f e g, em seguida, calcula-se o valor de *ranking* δ_{s_k}. O indivíduo somente será inserido na população se estiver bem adaptado ($\delta_{s_k} > \tau$).

O ATP trabalha com o conceito de população dinâmica, sendo o tamanho da população controlado pelo limiar de rejeição τ. A cada geração o valor de τ é incrementado e os indivíduos menos adaptados ($\delta_{s_k} \leq \tau$) são eliminados da população.

O processo evolutivo do ATP é realizado por um certo número de gerações ($MAX_{geracoes}$) ou enquanto a população não estiver vazia, ou seja, não existir mais indivíduos na população.

5.2.2 Componente AI

O componente AI realiza um agrupamento iterativo de cada indivíduo s_i selecionado para recombinação. Define-se inicialmente um número máximo de *clusters* 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 o indivíduo s_i e o centro de algum *cluster* for menor que o raio deste, a informação do indivíduo deve causar uma perturbação (assimilação) no centro do *cluster* mais similar. Caso contrário, a informação do indivíduo é considerada

suficientemente nova e esta é armazenada no centro de um novo *cluster*.

Neste trabalho utilizou-se a métrica 2-Troca para calcular a distância entre duas soluções. Esta métrica computa a quantidade de trocas necessárias para transformar o indivíduo s_i no centro do *cluster* c_i. A figura 5.3 mostra a métrica 2-Troca entre duas soluções.

s_k	3	5	-1	0	7	-4	6	-2	número de trocas
1)	**1**	5	**3**	0	7	-4	6	-2	1
2)	1	**7**	3	0	**5**	-4	6	-2	1
3)	1	7	**0**	**3**	5	-4	6	-2	1
4)	1	7	0	-3	**6**	-4	**5**	-2	1
5)	1	7	0	-3	6	**-2**	5	**4**	1
c_i	1	7	0	-3	6	-2	5	4	5

FIGURA 5.3 – Exemplo da métrica de distância 2-Troca.
FONTE: adaptada (Oliveira, 2004)

Um indivíduo será similar a um determinado *cluster* quando houver no mínimo 10% de coincidência de vértices entre as soluções. Sendo o raio do *cluster* assim definido:

$$r_t = \lceil 0,9n \rceil \tag{5.12}$$

onde n é o número de vértices do problema.

O processo de assimilação pode ser implementado de diferentes formas: assimilação simples, assimilação por recombinação e assimilação por caminho. As duas primeiras formas geram um único ponto interno (indivíduo) a ser avaliado, enquanto que na assimilação por caminho vários pontos são gerados e avaliados, sendo o melhor mantido como centro do *cluster*.

Optou-se por utilizar a assimilação por caminho, utilizando o método *path-relinking* (seção 4.2.4). Este método realiza movimentos exploratórios na trajetória que interconecta o indivíduo s_i e o centro do *cluster* 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*.

5.2.3 Componente AA

O componente AA é executado toda vez que um indivíduo 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_t dada pela equação 5.13, significando que um certo padrão de informação se tornou predominante no processo de evolução, sendo o centro deste *cluster* refinado através do componente AO.

$$\lambda_t \geq PD.\frac{NS}{|C_t|} \tag{5.13}$$

onde PD é o número de vezes que a densidade deve estar acima do normal para um *cluster* ser considerado promissor, NS é o número de indivíduos selecionados no intervalo entre a análise dos *clusters* não ativados e $|C_t|$ é o número de *clusters* existentes na geração t.

O componente AA também tem como função executar um esfriamento de todos os *clusters* que foram ativados a cada geração. Neste trabalho, o esfriamento coloca estes *clusters* em um estado chamado de *inativo*, sendo que estes podem se tornar novamente ativos, desde que aconteça uma melhora no centro durante o processo de assimilação. Outra função do AA é eliminar, a cada geração, os *clusters* com baixa densidade e os *clusters* que estão no estado inativo.

A eliminação dos *clusters* com baixa densidade permite que outros centros sejam descobertos. Segundo Oliveira (2004) pode-se fazer uma analogia aos ecossistemas naturais, onde os indivíduos em cada região interagem com o meio-ambiente, promovendo uma forma de exploração desse meio. Uma vez esgotados os recursos naturais de uma região, os indivíduos podem migrar para outras regiões ainda não exploradas.

5.2.4 Componente AO

No componente AO utiliza-se a heurística 2-Opt (seção 4.2.3), objetivando melhorar a solução presente no centro de um *cluster* promissor. O método implementado no AO é executado enquanto a heurística 2-Opt estiver conseguindo encontrar soluções melhores para o centro c_i. Caso AO encontre uma solução que seja melhor que o centro do *cluster*, este é atualizado com a solução encontrada.

A forma como a heurística 2-Opt foi implementada neste componente, faz com que o algoritmo de busca local seja um método de descida. Uma vez que, todos os possíveis vizinhos de uma solução são analisados, movendo-se somente para aquele que representar

uma melhora no valor atual da função objetivo. O método pára quando um ótimo local for encontrado.

5.3 *CS aplicado ao PCVCP

Outra abordagem proposta para resolver o PCVCP é o *CS, que propõe substituir o componente AE por uma outra metaheurística, sendo que esta precisa ser capaz de gerar um grande número de soluções diferentes para o processo de agrupamento. Neste trabalho propõe-se utilizar as metaheurísticas GRASP e VNS, gerando três abordagens diferentes. A primeira e a segunda abordagens, $^*CS_{grasp}$ e $^*CS_{vns}$, utilizam respectivamente GRASP e VNS como geradores de soluções para o processo de agrupamento. A terceira abordagem, $^*CS_{grasp-vns}$, combina GRASP e VNS afim de obter soluções iniciais com maior qualidade.

Para a primeira abordagem $^*CS_{grasp}$, optou-se por um GRASP um pouco mais sofisticado que o GRASP básico, conhecido na literatura como GRASP Reativo, Prais e Ribeiro (2000). Este procura combinar as boas características dos algoritmos puramente gulosos e dos procedimentos aleatórios, podendo gerar várias soluções diferentes. Tendo como vantagem o fato de que o parâmetro α é definido dinamicamente, permitindo a elaboração de listas de candidatos restritas (LCR) com tamanhos diferentes.

Na fase de construção do GRASP Reativo utilizou-se a função de economia da heurística *Add-step* (equação 4.1) para construir a lista dos elementos candidatos (C) a serem inseridos na solução. Cada elemento é selecionado de forma aleatória a partir de uma parte da lista C, contendo os melhores candidatos, chamada lista de candidatos restrita (LCR), sendo este elemento inserido na solução e a lista de candidatos atualizada. A fase de construção é executada enquanto existir candidatos com economia negativa ou a soma dos prêmios coletados for menor que o prêmio mínimo pré-estabelecido.

No GRASP Reativo ao invés de utilizar um valor fixo para α, propõe-se que α seja selecionado aleatoriamente a partir de um conjunto discreto $\psi = \{0.1, 0.2, ..., 1.0\}$. Desta forma, serão construídas listas de candidatos restritas diferentes a cada iteração, permitindo a geração de soluções diferentes que não seriam alcançadas através da utilização de único valor fixo.

Para cada valor de α do conjunto exite associado uma probabilidade $\rho_i(\alpha)$ deste ser escolhido, inicialmente todos têm a mesma probabilidade. Estas probabilidades precisam ser atualizadas periodicamente, utilizou-se a estratégia de qualificação absoluta (Prais e Ribeiro, 2000) nesta atualização.

Na estratégia de qualificação absoluta temos o valor da melhor solução encontrada até o momento (z^*) e o valor médio das soluções obtidas utilizando cada valor de α (A_i) na fase de construção. A distribuição de probabilidade é atualizada a cada 100 iterações realizadas, através da expressão 5.14.

$$\rho_i(\alpha) = \frac{q_i}{\sum_{j \in m} q_j} \tag{5.14}$$

onde $q_i = \frac{z^*}{A_i}$

Quanto mais adequado for um valor α_i, isto é, quanto menor for o valor de A_i, maior será a probabilidade de selecionar este valor de α, e consequêntemente, maior será a frequência com que este será utilizado na fase de construção.

Na fase de busca local do GRASP será utilizada a heurística 2-Troca, realizando trocas de vértices que fornecerem uma melhora no valor da função objetivo.

Para a segunda abordagem ${}^*CS_{vns}$ porpõe-se utilizar a metaheurística VNS. 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.

O VNS parte de uma solução inicial qualquer e procura refinar esta solução. Nesta abordagem, uma solução inicial será gerada aleatoriamente, mas garantindo que o prêmio mínimo seja coletado.

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

- $m_1 \Rightarrow$ Retirar vértices: escolhe-se dois vértices que façam parte da rota, e estes são retirados;
- $m_2 \Rightarrow$ Trocar vértices: escolhe-se quatro vértices que façam parte da rota e estes são trocados de posição aleatoriamente;
- $m_3 \Rightarrow$ Inserir vértices: escolhe-se dois vértices que não façam parte da rota, caso existam, e estes são inseridos em posições aleátorias na rota;

A partir da solução inicial gerada, a cada iteração seleciona-se aleatoriamente um vizinho s' na $k-$ésima vizinhança da solução corrente s, realizando-se o movimento m_k, onde $k = \{1, 2, 3\}$ definido anteriormente. Esse vizinho é então submetido a um procedimento de busca local. Se a solução ótima local, s'', for melhor que a solução s corrente, a busca

continua de s'' recomeçando da primeira vizinhança. Caso contrário, continua-se a busca a partir da próxima vizinhança. Este procedimento é encerrado quando o tempo sem melhora for maior que 1 segundo.

Na fase de busca local do VNS utiliza-se dois métodos de busca. O primeiro é o método *SeqAdd-SeqDrop* (Melo, 2001), que foi utilizado no ATP como heurística de treinamento, consistindo de uma sequência de inserções de vértices e uma sequência de remoções de vértices que melhorem a solução. O segundo método é a heurística 2-Opt, que procura pela melhor troca possível de 2 arestas da solução, realizando-a somente se esta produzir uma melhora no valor da função objetivo da solução.

A terceira abordagem ${}^*CS_{grasp-vns}$ consiste em combinar as duas primeiras apresentadas. Utilizando a fase de construção do GRASP Reativo para gerar uma solução inicial e o VNS como fase de busca local, procurando refinar a solução inicial encontrada. Espera-se que esta abordagem gere soluções melhores que as outras duas, e assim, possa ter uma convergência maior no processo de busca por agrupamento.

As abordagens *CS possuem os mesmos componentes que o ECS: agrupador iterativo (AI), analisador de agrupamentos (AA) e algoritmo de busca local (AO). Sendo que estes foram implementados de forma idêntica aos implementados no ECS, por isso não serão novamente descritos.

CAPÍTULO 6

RESULTADOS PRELIMINARES

As abordagens ECS e *CS para o PCVCP foram codificados em C++ e os experimentos foram conduzidos em uma máquina *AMD Athlon XP 1.53 GHz e memória de 256 MB*. Os experimentos foram realizados com objetivo de evidenciar a flexibilidade do método em relação ao algoritmo utilizado para alimentar o processo de agrupamento, e também para validar as abordagens propostas, mostrando que algoritmos de busca por agrupamentos podem ser competitivos para resolução do PCVCP.

O PCVCP não possui uma biblioteca pública de problemas testes, sendo assim, um conjunto de dez instâncias para o PCVCP encontradas em (Chaves et al., 2004b) serão utilizadas nestes experimentos. Essas instâncias têm diferentes números de vértices ($n \in \{10, 20, 30, 50, 100, 250, 500\}$), sendo todas geradas aleatoriamente dentro dos seguintes intervalos:

- custo de deslocamento entre os vértices: $c_{ij} \in [50, 1000]$;
- prêmio associado à cada vértice: $p_i \in [1, 100]$;
- penalidade associada à cada vértice: $\gamma_i \in [1, 750]$;

O prêmio mínimo, p_{mim}, a ser coletado representa 75% do somatório de todos os prêmios associados aos vértices.

A formulação matemática apresentada na seção 5.1 foi resolvida utilizando o *software* CPLEX versão 7.5, sendo os resultados obtidos apresentados na tabela 6.1. Na primeira coluna estão as instâncias do problema. Na coluna 2, $|V|$ representa a cardinalidade do conjunto de vértices que compõem a instância. A terceira e quarta colunas representam, respectivamente, o valor do ótimo global e o tempo de execução, em segundos.

TABELA 6.1 – Resultados do Cplex.

Instância	**|V|**	**ótimo global**	**tempo de execução (s)**	***gap***
v10	11	1765	0.06	0%
v20	21	2302	3.73	0%
v30a	31	3582	34.06	0%
v30b	31	2515	45.59	0%
v30c	31	3226	164.58	0%
v50a	51	4328	433439.97	0%
v50b	51	3872	241307.43	0%
v100a	101	6879	153059,09	2,46%

Nota-se que, através da formulação matemática proposta para o PCVCP neste trabalho, o CPLEX consegue resolver o problema para instâncias com até 30 vértices em um tempo computacional razoável. Entretanto, para as instâncias maiores, o problema já se torna inviável computacionalmente. Para as instâncias com 50 vértices, por exemplo, o CPLEX teve que ser executado por vários dias para poder encontrar o ótimo global. Uma instância com 100 vértices (*v100a*) foi executada utilizando o resultado encontrado em Chaves et al. (2004b) como valor de *upper bound*, permitindo acelerar o processo de busca do CPLEX. Mesmo assim, está foi executada por vários dias e não conseguiu fechar o *gap* entre *lower* e *upper bounds*. Porém, a solução viável obtida pelo CPLEX para esta instância foi melhor que a solução apresentada em Chaves et al. (2004b).

Os valores para os parâmetros de desempenho das abordagens ECS e *CS foram ajustados através de várias execuções e também baseados no trabalho de Oliveira (2004).

- número de indivíduos selecionados a cada geração $NS = 200$;
- número máximo de *clusters* $MC = 20$;
- pressão de densidade $PD = 2,5$ para o ECS e $PD = 1,5$ para o *CS;
- limite superior G_{max} é o valor da função f do pior indivíduo existente na população inicial;
- constante de equilíbrio $d = 1/G_{max}$;
- incremento do limiar de rejeição $\xi = 0,001$;
- tamanho inicial da população $|P_0| = 300$;
- número máximo de gerações $MAX_{geracoes} = 400$;
- percentual de indivíduos elite $\rho(e) = 20\%$;
- percentual de mutação sobre os novos indivíduos $\rho(m) = 30\%$;
- número de iterações do GRASP Reativo $NIter = 4000$;

Na tabela 6.2 são apresentados os resultados computacionais encontrados, até o presente momento, pelas abordagens ECS e *CS. Cada instância foi executada 10 vezes e a melhor solução obtida é apresentada, assim como o tempo necessário para encontrá-la.

Segundo a tabela 6.2, ECS e $^*CS_{grasp-vns}$ obtiveram os melhores resultados para as instâncias testadas, encontrando o ótimo global para as instâncias *v10, v20, v30a, v30b* e *v30c*. Entretanto, $^*CS_{grasp-vns}$ obteve resultados melhores para as instâncias

maiores. Com relação aos tempos de execução, nota-se que estes têm um aumento significativo à medida que o tamanho do problema aumenta. Tal fato pode ser explicado pela complexidade do PCVCP, e também pelo número de chamadas à função objetivo realizada pelo *path-relinking* no processo de assimilação, $((n+1).n)/2$.

TABELA 6.2 – Resultados Preliminares.

		Técnicas de solução							
		ECS		***CS**$_{grasp}$		***CS**$_{vns}$		***CS**$_{grasp-vns}$	
Instância	|V|	FO	Tempo	FO	Tempo	FO	Tempo	FO	Tempo
v10	11	1765	0.1	**1765**	**0.01**	**1765**	**0.01**	1765	0.09
v20	21	2302	7.75	2302	3.66	2302	6.05	**2302**	**1.33**
v30a	31	**3582**	**40.28**	3986	19.61	3736	30.47	3582	44.78
v30b	31	**2515**	**23.52**	2757	15.25	2596	23.93	2515	41.63
v30c	31	3236	59.56	3611	25.60	3355	53.29	**3236**	**27.92**
v50a	51	4898	612.54	5274	71.33	5047	315.60	**4680**	**576.11**
v50b	51	4512	489.68	4881	80.71	4732	307.85	**4175**	**527.78**
v100a	101	8320	1345.21	8805	527.58	9737	597.01	**8125**	**1053.36**
v250a	251	19342	4202.50	19953	1312.86	23705	3442.84	**18976**	**1363.33**
v500a	501	33535	8532.03	35495	6880.17	41059	6619.28	**33328**	**5195.65**

A tabela 6.3 apresenta uma comparação entre os resultados encontrados pelo ATP e pelo ECS, sendo que, o ECS utiliza o ATP como gerador de soluções para o processo de agrupamento. Observa-se que a inclusão do método de busca por agrupamentos do ECS sempre provoca uma melhora na solução obtida pelo ATP, ou seja, com o ECS têm-se uma convergência maior para as instâncias testadas.

TABELA 6.3 – Comparação entre ATP e ECS.

Instância	ATP	ECS	% de melhora
v10	1765	1765	0
v20	2336	2302	1,48
v30a	4086	3582	14,07
v30b	2780	2515	10,54
v30c	3605	3236	11,40
v50a	5867	4898	19,78
v50b	5275	4512	16,91
v100a	9783	8320	17,58
v250a	26165	19342	35,22
v500a	40521	33535	20,83

Na tabela 6.4 é mostrada uma comparação entre as três abordagens propostas para o *CS e os resultados obtidos utilizando apenas o método gerador de soluções, no caso, GRASP, VNS e GRASP-VNS. Nesta tabela observa-se que as abordagens $^*CS_{grasp}$ e $^*CS_{vns}$ também produzem uma melhora em relação às metaheurísticas aplicadas sozinhas, entretanto, não foram suficientes para encontrar boas soluções. Observa-se também que na abordagem $^*CS_{grasp-vns}$, a melhora é bem menor do que nas outras abordagens. As prováveis causas disto são: a) as soluções geradas pelo método GRASP-VNS são bons ótimos locais, às vezes ótimos globais, e isto não permite que estas soluções sejam melhoradas no processo de agrupamento; ou b) o método pode estar dirigindo a busca para uma pequena amostra do espaço de busca, prejudicando a detecção de regiões promissoras pelo processo de agrupamento.

TABELA 6.4 – Comparação entre GRASP, VNS e GRASP-VNS e as abordagens *CS.

Instância	Grasp	$^*CS_{grasp}$	%	VNS	$^*CS_{vns}$	%	GraspVns	$^*CS_{g.-vns}$	%
v10	1765	1765	0	1765	1765	0	1765	1765	0
v20	2559	2302	11,16	2302	2302	0	2302	2302	0
v30a	4148	3986	4,06	3836	3736	2,68	3685	3582	2,88
v30b	3074	2757	11,50	2717	2596	4,66	2567	2515	2,07
v30c	3733	3611	3,38	3705	3355	10,43	3284	3236	1,48
v50a	5386	5274	2,12	5969	5047	18,27	4907	4680	4,85
v50b	5184	4881	6,21	5357	4732	13,21	4219	4175	1,05
v100a	9106	8805	3,42	10455	9737	7,37	8566	8125	5,43
v250a	20351	19953	1,99	24716	23705	4,26	20183	18976	6,36
v500a	36394	35495	2,53	45570	41059	10,97	33901	33328	1,72

A tabela 6.5 apresenta os resultados obtidos através da execução do método proposto em Chaves et al. (2004b) utilizando o conceito de metaheurística híbrida, combinando GRASP e VNS/VND. Mostra-se também uma comparação com os resultados preliminares obtidos neste trabalho, através das abordagens ECS e $^*CS_{grasp-vns}$, uma vez que esta última obteve os melhores resultados para as abordagens *CS.

Observando a tabela 6.5, o método proposto em Chaves et al. (2004b) obteve resultados melhores para as maiores instâncias e o tempo computacional também foi menor praticamente em todos os testes. Porém, podemos identificar que os métodos de busca por agrupamentos também podem ter desempenhos satisfatórios, necessitando ainda uma melhora no tempo de convergência, assim como, uma melhora no próprio processo de convergência.

TABELA 6.5 – Comparação entre GRASP-VNS/VND, ECS e *CS$_{grasp-vns}$.

	GRASP-VNS/VND		ECS		*CS$_{grasp-vns}$	
Instância	FO	tempo	FO	tempo	FO	tempo
v10	**1765**	**0,1**	1765	0.1	1765	0.09
v20	**2302**	**1,04**	2302	7.75	2302	1.33
v30a	**3582**	**5,43**	3582	40.28	3582	44.78
v30b	**2515**	**3,83**	2515	23.52	2515	41.63
v30c	**3236**	**7,83**	3236	59.56	3236	27.92
v50a	**4328**	**332,45**	4898	612.54	4680	576.11
v50b	**3872**	**243,76**	4512	489.68	4175	527.78
v100a	**6892**	**892,09**	8320	1345.21	8125	1053.36
v250a	**15310**	**1118,33**	19342	4202.50	18976	1363.33
v500a	**28563**	**2345,79**	33535	8532.03	33328	5195.65

A implementação das metaheurísticas GRASP e VNS na abordagem *CS$_{grasp-vns}$ se difere da apresentada em Chaves et al. (2004b) em vários aspectos. Chaves et al. (2004b) utiliza apenas uma iteração para o GRASP, aplicando um filtro na fase de construção, sendo que, apenas a melhor solução construída é refinada através do método VNS/VND. Neste trabalho utiliza-se o GRASP Reativo, o qual não utiliza um valor fixo para o parâmetro α, procurando analisar mais amplamente o espaço de busca. Utiliza-se também um número alto de iterações (4000), gerando um número significativo de soluções para que as regiões promissoras cubram o maior espaço possível do espaço de busca.

Visto que o método GRASP-VNS/VND (Chaves et al., 2004b) obteve os melhores resultados para o PCVCP, propõe-se uma nova abordagem *CS, na qual, será aplicado o algoritmo de busca através de agrupamentos no GRASP-VNS/VND. Uma solução inicial será construída utilizando GRASP e refinada através do VNS/VND. Simultaneamente, o processo de agrupamento será aplicado em cada vizinho gerado pelo VNS, sendo que, o VNS tende a gerar soluções cada vez mais distantes da solução inicial.

Algumas estatísticas importantes que foram colhidas nestes experimentos, dizem respeito à eficiência do componente de otimização (AO) e também ao processo de assimilação (*path-relinking*). Considerando uma execução típica do ECS, o componente AO foi chamado 1311 vezes a cada execução, tendo conseguido melhorar a solução em 46,30% das vezes. O *path-relinking* é executado quando um indivíduo selecionado para o processo de reprodução for similar à algum *cluster*, sendo chamado 37520 vezes durante o processo evolutivo e obtendo melhora em 14,29% dos casos.

Entretanto, as estatísticas para a abordagem *CS que obteve os melhores resultados

($^*CS_{grasp-vns}$) não foram muito boas. O número de chamadas ao AO foi pequena (153 vezes), conseguindo melhorar a solução somente 13,72% dos casos. O *path-relinking* também não teve um desempenho satisfatório, tendo conseguido encontrar soluções melhores em apenas 10,6% das 3337 vezes em que foi executado.

Algumas considerações com relação ao ATP podem ser observadas nas figuras 6.1 e 6.2. A primeira figura mostra o tamanho da população ao longo das gerações. Pode-se observar que, assim como era esperado, a população tende a crescer no início, aceitando a maioria dos novos indivíduos gerados. Com o passar das gerações, o aumento do valor de τ determina uma quantia cada vez maior de indivíduos a serem eliminados, terminando o processo evolutivo com uma pequena quantidade de indivíduos.

O gráfico de convergência do ATP pode ser observado na figura 6.2. O gráfico mostra que a população converge para a heurística de treinamento ao longo do processo evolutivo, assumindo as características desta heurística. Entretanto, através dos resultados mostrados na tabela 6.3, podemos observar que a heurística de treinamento empregada não foi suficiente para conduzir o ATP a boas soluções.

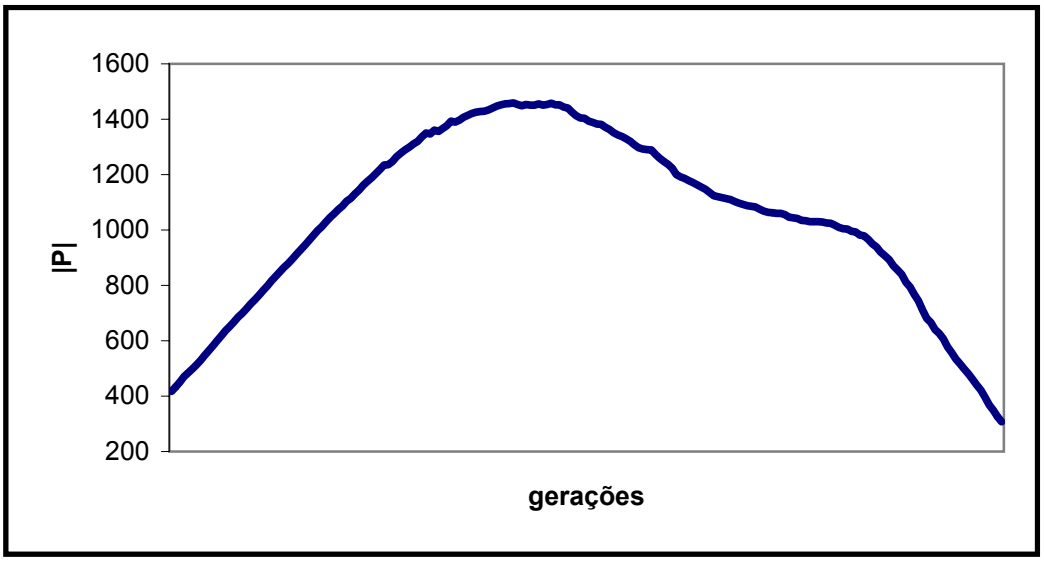

FIGURA 6.1 – Comportamento da população do ATP para a instância *v30a*.

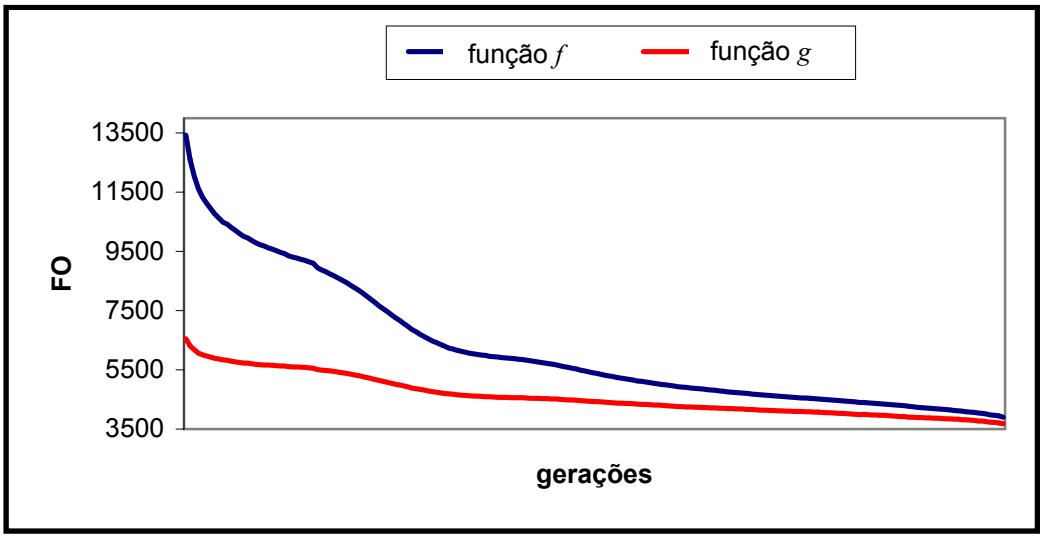

FIGURA 6.2 – Convergência do ATP para a instância *v30a*.

CAPÍTULO 7

CONSIDERAÇÕES FINAIS

Este trabalho propõe duas abordagens para a resolução do PCVCP, ECS e *CS. Estas utilizam o conceito de algoritmos híbridos, combinando metaheurísticas com um processo de agrupamento de soluções em subespaços 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.

Além do processo de busca em *clusters* promissores, o próprio processo de agrupamento realiza operações de intensificação de busca, através do método *path-relinking*, dentro da região limitada pelo *cluster*.

A proposta deste trabalho se justifica segundo alguns aspectos. Primeiramente, pelo fato do ECS ter obtido sucesso ao ser aplicado a problemas com espaços de busca contínuos e discretos. Aumentando a eficiência da busca local durante o processo evolutivo e também apresentando desempenho superior com relação a outros métodos evolutivos propostos em Oliveira (2004) e alguns existentes na literatura.

Além disso, a substituição do componente AE por uma outra metaheurística, neste caso GRASP e VNS, contribui para a criação de uma nova abordagem, o $^*CS$, visando investigar a geração contínua de soluções diretamente para o processo de assimilação.

Os resultados preliminares obtidos, apesar de não serem melhores que os encontrados na literatura, mostram que estas abordagens podem ser competitivas, havendo ainda a necessidade de aprimoramentos, principalmente no que diz respeito à qualidade da solução e também com relação ao tempo de execução.

Com relação a continuação deste trabalho, propõe-se realizar uma análise dos parâmetros de desempenho das abordagens ECS e $^*CS$, assim como um estudo mais detalhado com relação ao comportamento destes métodos.

Propõe-se também a implementação de novas heurísticas de busca local, procurando melhorar a convêrgencia e o tempo de execução destes métodos. E ainda, substituir o componente AE por outras metaheurísticas, como Busca Tabu, ou por uma relaxação lagrangeana, além de aplicar o *CS em um método baseado no proposto em Chaves et al. (2004b).

Uma outra proposta, é resolver a formulação matemática para as instâncias maiores, utilizando uma nova versão do CPLEX (versão 9.0). Objetivando encontrar o ótimo global para esta instâncias ou obter *lower* e *upper bounds* que permitam uma comparação com os resultados obtidos nas abordagens heurísticas.

Deseja-se também analisar o comportamento destas abordagens para outros problemas testes existentes na literatura, caso seja possível encontrar. Assim como, modificar a percentagem do prêmio mínimo p_{min} a ser coletado.

O cronograma de execução da sequência dos trabalhos é apresentado na Tabela 7.1.

TABELA 7.1 – Cronograma de execução.

	2005							2006	
Fase	jun	jul	ago	set	out	nov	dez	jan	fev
1	•	•							
2		•	•	•					
3			•	•	•	•			
4				•	•				
5					•	•			
6					•	•	•		
7	•					•	•	•	
8									•

1) Análise e refinamento dos parâmetros de desempenho;

2) Implementação de novas heurísticas de busca local, procurando melhorar o desempenho das abordagens propostas;

3) Substituição do componente AE por outras metaheurísticas ou por uma relaxação lagrangeana;

4) Aplicação do *CS em um método baseado no proposto em Chaves et al. (2004b);

5) Resolução da formulação matemática para instâncias maiores, utilizando o *software* CPLEX versão 9.0;

6) Estudo de novos problemas testes, utilizando novas instâncias e também variando a percentagem de prêmio mínimo, p_{min}, as ser coletado;

7) Escrita da dissertação;

8) Entrega e defesa da dissertação;

REFERÊNCIAS BIBLIOGRÁFICAS

Balas, E. The Prize Collecting Travelling Salesman Problem. **Networks**, v. 19, p. 621–636, 1989. 19, 21, 25, 26, 45

Balas, E.; Martin, G. **ROLL-A-ROUND: Software Package for Scheduling the Rounds of a Rolling Mill**. 104 p. ©Copyright by Balas and Martin Associates, Pittsburgh, 1985. 21, 25

Bar-Yehuda, R.; Even, G.; Shahar, S. On Approximating a Geometric Prize-Collecting Traveling Salesman Problem with Time Windows. **Lecture Notes in Computer Science**, v. 2832, p. 55–66, 2003. 28

Bienstock, D.; Goemans, M. X.; Simchi-Levi, D.; Williamson, D. A note on the Prize Collecting Travelling Salesman Problem. **Mathematical Programming**, v. 59, n. 3, p. 413–420, 1993. 25

Canuto, S.; Resende, M.; Ribeiro, C. Local search with perturbations for the prize-collecting Steiner Tree Problem in graphs. **Networks**, v. 38, p. 50–58, 2001. 27

Chaves, A. A.; Biajoli, F. L.; Mine, O. M.; Souza, M. J. F. **Modelagens Exata e Heurística para Resolucao do Problema do Caixeiro Viajante com Coleta de Prêmios**. Ouro Preto: Departamento de Ciência da Computação - Universidade Federal de Ouro Preto, 2003. 47 p. 27, 36

——. Modelagens Exata e Heurística para Resolucao do Problema do Caixeiro Viajante com Coleta de Prêmios. In: Encontro Nacional de Engenharia de Produção (ENEGEP), 24., 2004, Florianópolis. **Anais...** Rio de Janeiro: ENEGEP, 2004a. p. 3151–3158. 1 CD-ROM. 27

——. Modelagens Exata e Heurística para Resolução de uma Generalização do Problema do Caixeiro Viajante. In: Simpósio Brasileiro de Pesquisa Operacional (SBPO), 36., 2004, São João Del Rei. **Anais...** Rio de Janeiro: SOBRAPO, 2004b. p. 1367–1378. 1 CD-ROM. 27, 55, 56, 58, 59, 63, 64

Christofides, N. **Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem**. Pittsburg: School of Industrial Administration - Carnegie-Mellon University, 1976. 388 p. 25

Croes, G. A method for solving travelling salesman problems. **Operations Research**, v. 6, p. 791–812, 1958. 32

Dell'Amico, M.; Maffioli, F.; Sciomanchen, A. A Lagrangian Heuristic for the Prize Collecting Travelling Salesman Problem. **Anals of Operations Research**, v. 81, p. 289–305, 1998. 25, 26, 30

Dell'Amico, M.; Maffioli, F.; Värbrand, P. On Prize Collecting Tours and the Asymetric Travelling Salesman Problem. **International Transactions in Operational Research**, v. 2, n. 3, p. 297–308, 1995. 25

Dorigo, M.; Maniezzo, V.; Colorni, A. The Ant System: Optimization by a colony of cooperating agents. **IEEE Transactions on Systems, Man, and Cybernetics-Part B**, v. 26, n. 1, p. 29–41, 1996. 18

Feo, T.; Resende, M. Greedy randomized adaptive search procedures. **Journal of Global Optimization**, v. 6, p. 109–133, 1995. 18, 34, 35

Fischetti, M.; Toth, P. An Additive Approach for the Optimal Solution of the Prize Collecting Traveling Salesman Problem. In: Golden, B. L.; Assad, A. A. (eds.) **Vehicle Routing: Methods and Studies**. North-Holland: Elsevier Science, 1988. p. 319–343. 17, 25

Glover, F. Future Paths for Integer Programming and links to Artificial Intelligence. **Computers and Operations Research**, v. 5, p. 553–549, 1986. 18

——. Tabu search and adaptive memory programing: Advances, applications and challenges. In: Barr, R.; Helgason, R.; Kennington, J. (eds.) **Interfaces in Computer Science and Operations Research**. Kluwer, 1996. p. 1–75. 33

Goemans, M. X.; Williamson, D. P. A General Aproximation Technique for Constrained Forest Problems. **SIAM Journal on Computing**, v. 24, n. 2, p. 296–317, 1995. 25

Goldberg, D. E. **Genetic Algorithms in Search. Optimization and Machine Learning**. Addison-Wesley: Berkeley, 1989. 223 p. 18, 37

Gomes, L. M.; Diniz, V. B.; Martinhon, C. A. An Hybrid GRASP+VND Metaheuristic fo the Prize Collecting Traveling Salesman Problem. In: Simpósio Brasileiro de Pesquisa Operacional (SBPO), 32., 2000, Viçosa. **Anais...** Rio de Janeiro: SOBRAPO, 2000. p. 1657–1665. 1 CD-ROM. 26, 30, 31

Holland, J. H. **Adaptation in natural and artificial systems**. Ann Arbor: University of Michigan Press, 1975. 211 p. 37

ILOG CPLEX 7.5 Reference Manual. 7.5v. 610 p. ©Copyright by ILOG, France, 2001. 19, 47

Kirkpatrick, S.; Gellat, D.; Vecchi, M. Optimization by Simulated Annealing. **Science**, v. 220, p. 671–680, 1983. 18

Lopez, L.; Carter, M. W.; Gendreau, M. The hot strip mill production scheduling problem: A tabu search approach. **European Journal of Operational Research**, v. 106, p. 317–335, 1998. 26

Lorena, L.; Furtado, J. Constructive genetic algorithm for clustering problems. **Evolutionary Computation**, v. 9, n. 3, p. 309–327, 2001. 48

Lorena, L. A. N.; Lopes, L. S. Computational Experiments with Genetic Algorithms Applied to Set Covering Problems. **Pesquisa Operacional**, v. 16, n. 1, p. 41–53, Junho 1996. 37

Melo, V. A. **Metaheurísticas para o Problema do Caixeiro Viajante com Coleta de Prêmios**. 2001. 153 p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Computação, Universidade Federal Fluminense, Nitéroi. 2001. 26, 49, 54

Melo, V. A.; Martinhon, C. A. Metaheurísticas Híbridas para o Problema do Caixeiro Viajante com Coleta de Prêmios. In: Simpósio Brasileiro de Pesquisa Operacional (SBPO), 36., 2004, São João Del Rei. **Anais...** Rio de Janeiro: SOBRAPO, 2004. p. 1295–1306. 1 CD-ROM. 26

Mladenovic, N.; Hansen, P. Variable Neighborhood Search. **Computers and Operations Research**, v. 24, p. 1097–1100, 1997. 18, 36

Oliveira, A. C. M. **Algoritmos Evolutivos Híbridos com Detecção de Regiões Promissoras em Espaços de Busca Contínuo e Discreto**. 2004. 200 p. Tese (Doutorado em Computação Aplicada) – Instituto Nacional de Pesquisa Operacional (INPE), São José dos Campos. 2004. 18, 32, 37, 38, 39, 42, 48, 50, 51, 56, 63

Oliveira, A. C. M.; Lorena, L. A. N. Detecting promising areas by evolutionary clustering search. In: Bazzan, A. L. C.; Labidi, S. (eds.) **Advances in Artificial Intelligence**. Springer Lecture Notes in Artificial Intelligence Series, 2004. p. 385–394. 39

——. Population training heuristics. **Lecture Notes in Computer Science**, v. 3448, p. 166–176, 2005. 37

Pekny, J.; Miller, D. An exact parallel algorithm for the Resource Constrained Travelling Salesman Problem with Application to Scheduling with an Aggregate Deadline. In: ACM Annual Computer Science Conference, 18., 1990, Washington. **Anais...** New York: ACM Press, 1990. p. 208–214. 27

Prado, D. **Programação Linear**. Belo Horizonte: EDG, 1999. 205 p. 29

Prais, M.; Ribeiro, C. C. Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment. **INFORMS Journal on Computing**, v. 12, n. 3, p. 164–176, Summer 2000. 35, 52

Reinelt, G. **Traveling Salesman : Computational Solutions for TSP Aplications**. Springer: Lecture Notes in Computer Science, 1994. 223 p. 21

Syswerda, G. Uniform crossover in genetic algorithms. In: International Conference on Genetic Algorithms(ICGA), 3., 1989, Virginia. **Fairfax: Morgan Kaufmann...** George Mason University: Proceedings, 1989. p. 2–9. 48

Torres, R. D.; Brito, J. A. M. Problemas de Coleta de Prêmios Seletiva. In: Simpósio Brasileiro de Pesquisa Operacional (SBPO), 35., 2003, Natal. **Anais...** Rio de Janeiro: SOBRAPO, 2003. p. 1359–1371. 1 CD-ROM. 19, 26, 45

Tsiligirides, T. Heuristic Methods Applied to Orienteering. **Journal of Operational Research Society**, v. 35, n. 9, p. 797–809, 1984. 27