Sistemas & Gestão 1 (2) 116-131
Programa de Pós-graduação em Sistemas de Gestão, TEP/TCE/CTC/PROPP/UFF

Um Algoritmo Evolutivo para o Problema de Roteamento em Arcos Capacitados

Dalessandro Soares Vianna[1], dalessandro@ucam-campos.br

Roberta Claudino Barreto Pessanha Gomes[1], roberta.claudino@gmail.com

[1] Universidade Candido Mendes – UCAM-Campos

Núcleo de Pesquisa e Desenvolvimento em Informática - NPDI,

Campos dos Goytazazes, RJ 28040-320, Brasil.

**Recebido: Março, 2006 / Aceito: Maio, 2006*

RESUMO

O problema de roteamento em arcos capacitados (Capacitated Arc Routing Problem - CARP) consiste em visitar um subconjunto de arestas do grafo que descreve o problema, atendendo às suas demandas. Aplicações possíveis para o CARP são a coleta de lixo urbano e a inspeção de linhas de força. O CARP é um problema NP-difícil, mesmo no caso onde existe apenas um veículo (chamado de Problema do Carteiro Rural). Neste caso, o uso de metaheurísticas surge como uma estratégia de solução eficiente.Este trabalho apresenta um algoritmo genético híbrido para o CARP, que é testado em instâncias disponíveis na literatura. Os resultados obtidos até o momento demonstram a eficiência do algoritmo proposto quando comparado com limites inferiores descritos na literatura.

Palavras-Chave: Roteamento de veículos. Arcos capacitados. Algoritmos genéticos.

1. INTRODUÇÃO

O clássico Problema de Roteamento de Veículos (PRV) é definido da seguinte maneira: veículos com uma capacidade fixa Q devem sair do depósito (i = 0) e atender demandas q_i (i =1, ..., n) de n clientes. Conhecendo a distância d_{ij} entre os clientes i e j (i, j = 0, ..., n), o objetivo é minimizar a distância total percorrida pelos veículos, de tal maneira que apenas um veículo atenda a demanda de um dado cliente e que a capacidade Q dos veículos não seja violada.

Ao contrário do PRV, no qual os n nós possuem demandas a serem atendidas, o problema de roteamento em arcos capacitados (*Capacitated Arc Routing Problem* - CARP) consiste na visita de um subconjunto $|A|$ de arestas. Como possíveis aplicações para este problema têm-se a coleta de lixo urbano e a inspeção de linhas de força. A partir de agora, neste texto, os exemplos serão baseados na coleta de lixo urbano.

Na literatura, o CARP básico utiliza redes não-direcionadas. Cada aresta modela uma rua de sentido duplo, em que ambos os lados são tratados em paralelo e em qualquer direção (coleta *zigzag*), uma prática comum em áreas residenciais com ruas estreitas. Uma frota de veículos idênticos de capacidade limitada *Q* tem como base um depósito. Cada aresta pode ser percorrida várias vezes, com um custo de travessia pré-determinado. Algumas arestas são *requeridas*, isto é, possuem uma demanda (quantidade de lixo) diferente de zero para ser coletada por um veículo. O CARP consiste em determinar um conjunto de viagens dos veículos, totalizando um custo mínimo. Cada uma destas viagens deve começar e terminar no depósito. Cada aresta requerida é atendida por uma única viagem. A demanda total atendida por uma viagem não pode ultrapassar *Q*.

O CARP é um problema NP-difícil, mesmo no caso onde existe apenas um veículo (chamado de problema do carteiro rural - PCR). Uma vez que métodos exatos estão ainda limitados a problemas com 20-30 arestas [Hirabayashi *et al.*, 1992], heurísticas são necessárias para resolver instâncias maiores [Golden & Wong, 1981; Golden *et al.*, 1983; Pearn, 1991].

Existem algumas metaheurísticas definidas para o CARP: um algoritmo baseado na técnica *simulated annealing* [Eglese, 1994]; algoritmos de busca tabu para o CARP [Eglese & Li, 1996; Hertz *et al.*, 2000] ou para casos particulares como o PCR não-direcionado [Hertz *et al.*, 1999] ou o PCR misto [Corberán *et al.*, 2000].

Este trabalho propõe um algoritmo genético híbrido para o CARP. Os resultados obtidos até o momento demonstram a eficiência do algoritmo proposto quando comparado com os limites inferiores descritos em [Belenguer & Benavent, 2003].

O restante deste artigo é organizado da seguinte maneira. Na Seção 2 será apresentado, com maiores detalhes, o algoritmo genético híbrido proposto. Os resultados computacionais são apresentados na Seção 3. Por fim, são apresentadas as conclusões.

2. ALGORITMO GENÉTICO HÍBRIDO

Métodos baseados em computação evolutiva constituem uma classe de algoritmos de busca e otimização estocástica inspirados na teoria da evolução natural de Darwin. Estes algoritmos têm recebido especial atenção nos últimos tempos por se tratarem de métodos robustos, capazes de fornecer soluções de alta qualidade para problemas considerados intratáveis por métodos tradicionais de otimização, os quais foram concebidos para problemas lineares, contínuos e diferenciáveis. Mas, como é observado em [Schwefel, 1994], o mundo real é não-linear e dinâmico, cheio de fenômenos como descontinuidade, instabilidade estrutural e formas geométricas fractais. Em problemas em que precisamos levar em conta tais fenômenos, os métodos tradicionais certamente não apresentarão desempenho satisfatório. Métodos evolutivos (algoritmos genéticos, programação genética, estratégias evolutivas e programação evolutiva) são uma alternativa para tentar superar as limitações apresentadas por métodos tradicionais, embora não garantam a obtenção da solução exata.

Os sistemas baseados em computação evolutiva mantêm uma população de soluções potenciais, aplicam processos de seleção baseados na adaptação de um indivíduo e também empregam outros operadores “genéticos”. Diversas abordagens para sistemas baseados em evolução foram propostas, sendo que as principais diferenças entre elas dizem respeito aos operadores genéticos empregados.

Umas das principais abordagens para sistemas baseados em evolução são os algoritmos genéticos (AGs). Os AGs foram introduzidos por Holland (1992) com o objetivo de formalizar matematicamente e explicar processos de adaptação em sistemas naturais e desenvolver sistemas artificiais (simulados em computador) que retenham os mecanismos originais encontrados em sistemas naturais. Os algoritmos genéticos têm sido intensamente aplicados em problemas de otimização [Drummond *et al*., 2001; Ochi *et al*., 1998a; Ochi *et*

al., 1998b; Vianna *et al*., 1999], apesar de não ter sido este o propósito original que levou ao seu desenvolvimento.

Um algoritmo genético é um algoritmo probabilístico que mantém uma população de indivíduos $Pop(t) = \{x_1^t, ..., xP^t_{tampop}\}$ na iteração (geração) t. Cada indivíduo representa um candidato à solução do problema em questão e, em qualquer implementação computacional, assume a forma de alguma estrutura de dados. Cada solução x_i^t é avaliada e produz alguma medida de adaptação (*fitness*). Alguns indivíduos da população são submetidos a transformações (passo de alteração) por meio de operadores genéticos para formar novas soluções. Existem transformações unárias (mutação) que criam novos indivíduos através de pequenas mudanças em um indivíduo e transformações de ordem superior (cruzamento) que criam novos indivíduos através da combinação de dois ou mais indivíduos. Uma nova população (geração t+1) é formada pela seleção dos indivíduos mais adaptados (passo de seleção) da geração t. Após um número de gerações, a condição de parada deve ser atendida, a qual geralmente indica a existência, na população, de um indivíduo que represente uma solução quase-ótima (de boa qualidade) para o problema.

Os algoritmos genéticos empregam uma terminologia originada da teoria da evolução natural e da genética. Um indivíduo da população é representado por um único cromossomo, o qual contém a codificação (genótipo) de uma possível solução do problema (fenótipo). Cromossomos são usualmente implementados na forma de vetores, onde cada componente do vetor é conhecido como gene. Os possíveis valores que um determinado gene pode assumir são denominados alelos.

O processo de evolução executado por um algoritmo genético corresponde a um processo de busca em um espaço de soluções potenciais para o problema. Como enfatiza [Michalewicz, 1996], esta busca requer um equilíbrio entre dois objetivos aparentemente conflitantes: o aproveitamento das melhores soluções e a exploração do espaço de busca (explotação $\times$ exploração). Métodos de otimização do tipo *hill climbing* são exemplos de métodos que aproveitam a melhor solução na busca de possíveis aprimoramentos; em compensação, estes métodos ignoram a exploração do espaço de busca. Métodos de busca aleatória são exemplos típicos de métodos que exploram o espaço de busca ignorando o aproveitamento de regiões promissoras do espaço. Algoritmos genéticos constituem uma classe de métodos de busca de propósito geral que apresentam um balanço entre aproveitamento de melhores soluções e exploração do espaço de busca.

Os algoritmos genéticos pertencem à classe dos algoritmos probabilísticos, mas não são métodos de busca puramente aleatórios, pois combinam elementos de métodos de busca diretos e estocásticos. Outra propriedade importante dos algoritmos genéticos é que eles mantêm uma população de soluções candidatas, enquanto que os métodos alternativos, como *simulated annealing*, processam um único ponto no espaço de busca a cada instante.

O processo de busca realizado pelos algoritmos genéticos é multi-direcional, através da manutenção de soluções candidatas, e encoraja a troca de informação entre as direções. A cada geração, soluções relativamente “boas" se reproduzem, enquanto que soluções relativamente “ruins" são eliminadas. Para fazer a distinção entre diferentes soluções é empregada uma função objetivo (de avaliação ou de adaptabilidade) que simula o papel da pressão exercida pelo ambiente sobre o indivíduo.

Um algoritmo genético para um problema particular deve ter os seguintes componentes:

- uma representação genética para soluções candidatas ou potenciais (processo de codificação);
- uma maneira de criar uma população inicial de soluções candidatas ou potenciais;

- uma função de avaliação que faz o papel da pressão ambiental, classificando as soluções em termos de sua adaptação ao ambiente (ou seja, sua capacidade de resolver o problema);
- operadores genéticos;
- um método de seleção de indivíduos;
- valores para os diversos parâmetros usados pelo algoritmo genético (tamanho da população, probabilidades de aplicação dos operadores genéticos, etc.).

O sucesso do uso de AGs híbridos, que incorporam ferramentas ou técnicas de busca local de outras metaheurísticas, pode ser notado em diversos trabalhos da literatura científica [Glover, 1994; Glover, 1995; Rochat & Taillard, 1995].

Nas subseções seguintes serão apresentadas as etapas que compõem o algoritmo genético híbrido proposto para o CARP.

2.1. ALGORITMO DE CONSTRUÇÃO ALEATORIZADO

O algoritmo de construção utilizado é baseado no método do vizinho mais próximo [Cormen *et al*., 2002]. A Figura 1 apresenta o pseudo-código do algoritmo CONSTRUIRINDIVIDUO. Este algoritmo recebe como parâmetro de entrada o índice *H*, que define a aleatoriedade do algoritmo (neste trabalho utilizou-se *H*=3), e retorna como saída a solução *s* construída. Inicialmente, todas as arestas encontram-se desmarcadas, pois não houve atendimento a nenhuma demanda. O nó corrente (*Vatual*) neste momento é o depósito, de acordo com a linha 1, uma vez que é deste vértice que partem os veículos para coleta. Na linha 6, é feita uma ordenação crescente de todas as arestas que partem de *Vatual*, de acordo com o custo e dando prioridade às arestas desmarcadas. Se existirem arestas desmarcadas partindo de *Vaux*, na linha 14 é feito um sorteio entre as *H* primeiras arestas desmarcadas. Assim, é dada prioridade de escolha às arestas mais próximas de *Vaux*. Após a escolha da aresta (*Vatual*, *Vaux*), sendo esta desmarcada, verifica-se se é possível atender à sua demanda na linha 15 e em caso positivo, a aresta escolhida é inserida na rota na linha 17, soma-se seu custo e sua demanda nas linhas 18 e 19, respectivamente. A aresta é marcada na linha 20 e o novo nó corrente passa a ser *Vaux*, linha 21. Em caso de uma aresta desmarcada que não pode ser atendida, linha 22, esta aresta é descartada. Na linha 23 é feita a volta ao depósito através do caminho de menor custo (determinado pelo algoritmo Dijkstra [Cormen *et al*., 2002]). Na linha 24, a demanda do veículo é zerada e, na linha 25, o nó corrente volta a ser o depósito. Se não existirem arestas desmarcadas partindo de *Vatual*, linha 7, o sorteio é feito entre todas as arestas que partem de *Vatual* na linha 9. Neste caso, a aresta é inserida na linha 10, apenas seu custo é somado (linha 11) e *Vaux* passa a ser o nó corrente (linha 12).

```
Procedimento CONSTRUIRINDIVIDUO (H)
Entrada
    H - define a aleatoriedade do algoritmo;
Saída
    s - solução construída;
Início
01   Vatual ← 0 {que representa o depósito};
02   s.custo ← 0;
03  demanda_atual ← 0;
04  Enquanto existir aresta desmarcada faça
05  Início
06      Ordenar as arestas partindo de Vatual priorizando as arestas desmarcadas;
07      Se não existirem arestas desmarcadas partindo de Vatual então
08      Início
09          Vaux ← sorteio entre todas as arestas (Vatual, Vaux);
10          Inserir a aresta (Vatual, Vaux) na rota;
11          s.custo ← s.custo + custo de (Vatual, Vaux);
12          Vatual  ← Vaux;
13      Senão
14          Vaux ← sorteio entre as H primeiras arestas (Vatual, Vaux) desmarcadas,
                onde H é um parâmetro de entrada do algoritmo;
15          Se a demanda de (Vatual, Vaux) + demanda_atual  ≤ Q então
16          Início
17              Inserir a aresta (Vatual, Vaux) na rota;
18              s.custo ← s.custo + custo de (Vatual, Vaux);
19              demanda_atual  ← demanda_atual + demanda de (Vatual, Vaux);
20              Marcar a aresta (Vatual, Vaux);
21              Vatual  ← Vaux;
22          Senão
23              Inserir as arestas e seus respectivos custos que fazem parte do caminho
                    de menor custo de Vatual até o depósito;
24              demanda_atual ← 0;
25              Vatual ← 0;
26          Fim-Se
27      Fim-Se
28  Fim-Enquanto
29  Inserir as arestas e seus respectivos custos que fazem parte do caminho de menor
        custo de Vatual até o depósito;
30  Retorne s;
Fim-CONSTRUIRINDIVIDUO
```

Figura 1 – Algoritmo de construção aleatorizado.

Para ilustrar o processo de construção de soluções, tomou-se como exemplo o grafo correspondente a um dos problemas testes (gdb17) que serão mais bem discutidos na seção de resultados computacionais (Seção 3). A Figura 2 e a Tabela 1 apresentam as características deste grafo.

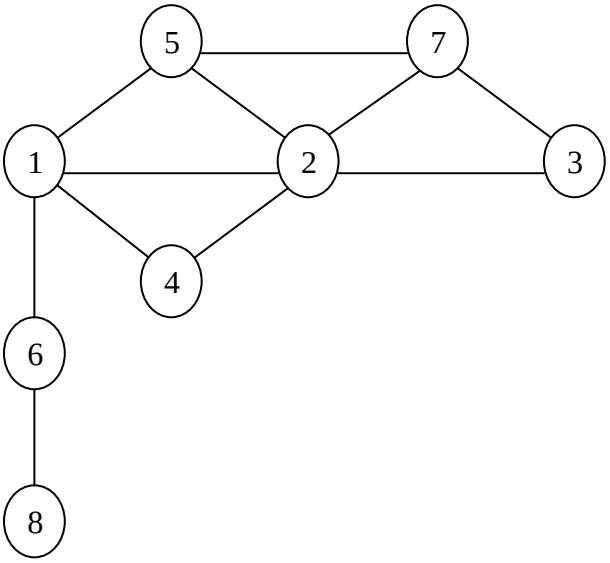

Figura 2 – Representação gráfica do problema teste gdb17.

Tabela 1 – Custos e demandas das aretas do problema teste gdb17.

Arestas	**Custo**	**Demanda**
1 - 2	4	8
1 - 4	3	3
1 - 5	1	5
1 - 6	2	8
2 - 3	1	4
2 - 4	9	6
2 - 5	5	1
2 - 7	2	9
3 - 7	6	8
5 - 7	7	9
6 - 8	5	5

Inicialmente, todas as arestas se encontram desmarcadas. Ou seja: nenhuma aresta teve sua demanda atendida. Os veículos, no problema teste gdb17, têm uma capacidade de 27 unidades e devem retornar ao depósito toda vez que seu limite for alcançado. Estes partem do depósito, representado pelo vértice 1. É feito um sorteio entre, no máximo, as *H*=3 arestas de menor custo – (1,5), (1,6), (1,4) – partindo do nó corrente (*Vatual*=1). Se, através do sorteio, a aresta (1,5) for escolhida, sua demanda é coletada, seu custo contabilizado e o nó corrente passa ser *Vatual*=5. A solução parcial, descrita abaixo, pode ser visualizada na Figura 3.

Solução: (1,5)

Custo: 1

Demanda Total: 5

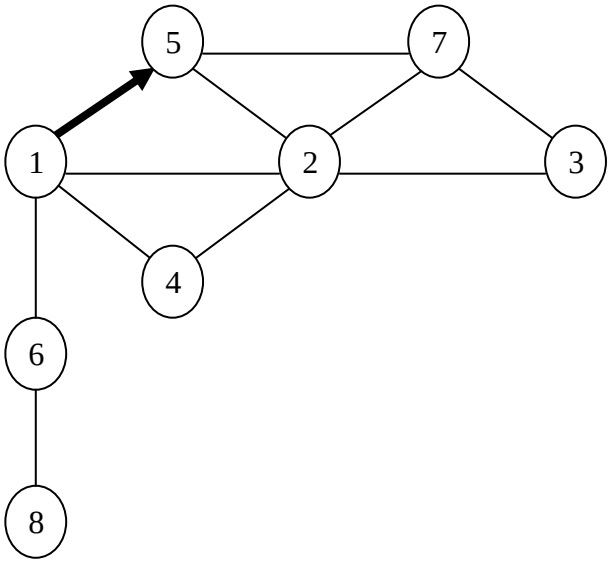

Figura 3 – Procedimento de Construção após a escolha da primeira aresta.

Novamente é feito um sorteio entre as, no máximo, *H*=3 arestas de menor custo – (5,1), (5,2) – partindo do nó corrente (*Vatual*=5), mas desta vez observando certos detalhes: a demanda da aresta escolhida mais a demanda já atendida não pode ultrapassar a capacidade do veículo; e deve-se priorizar as arestas ainda não atendidas. Dando continuidade a esse processo, a primeira rota é fechada após a escolha da quinta aresta, como é mostrado na solução a seguir e na Figura 4.

Solução: (1,5) (5,2) (2,3) (3,7) (7,5)

Custo: 20

Demanda Total: 27

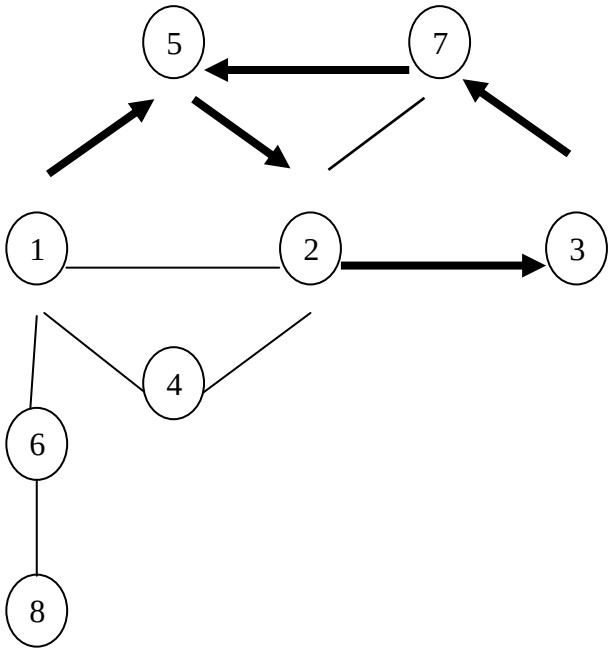

Figura 4 – Procedimento de Construção após o fechamento da primeira rota.

Uma vez que é impossível atender outras demandas, o veículo deve retornar ao depósito, escolhendo para isso o caminho de menor custo (algoritmo Dijkstra [Cormen *et al*., 2002]). Neste exemplo, o caminho de retorno ao depósito é formado apenas pela aresta (5,1). Esse processo deve ser repetido até que se gere uma quantidade de rotas suficiente para atender a todas as demandas. A solução final para o exemplo dado é descrita a seguir.

Solução: (1,5) (5,2) (2,3) (3,7) (7,5) (5,1) (1,4) (4,2) (2,7) (7,2) (2,1) (1,6) (6,8) (8,6) (6,1)

Custo: 55

2.2. FUNÇÃO OBJETIVO

O custo de uma rota *r* será o somatório dos custos de travessia associado a cada aresta pertencente a *r*. O valor objetivo ou custo de uma solução para o CARP será o somatório dos custos de todas as rotas da solução.

2.3. CRIAÇÃO DA POPULAÇÃO INICIAL

Na fase inicial do algoritmo genético, uma população de *tamPop* = 500 indivíduos é gerada através do algoritmo de construção aleatorizado CONSTRUIRINDIVIDUO descrito na Figura 1.

2.4. EVOLUÇÃO DA POPULAÇÃO

Para a evolução da população seguiu-se a estratégia proposta em [Bean, 1994] e já usada com sucesso em [Buriol *et al*., 2003; Ribeiro & Vianna, 2003]. Nesta estratégia, a evolução da população ao longo das gerações é realizada como mostrado na Figura 5. A cada geração *k*, a população é ordenada e dividida em classes. A classe *A* é composta pelos melhores indivíduos, que representam 25% da população. A classe *C* é formada pelos piores indivíduos, que representam 5% da população. A classe *B* é formada pelos indivíduos restantes.

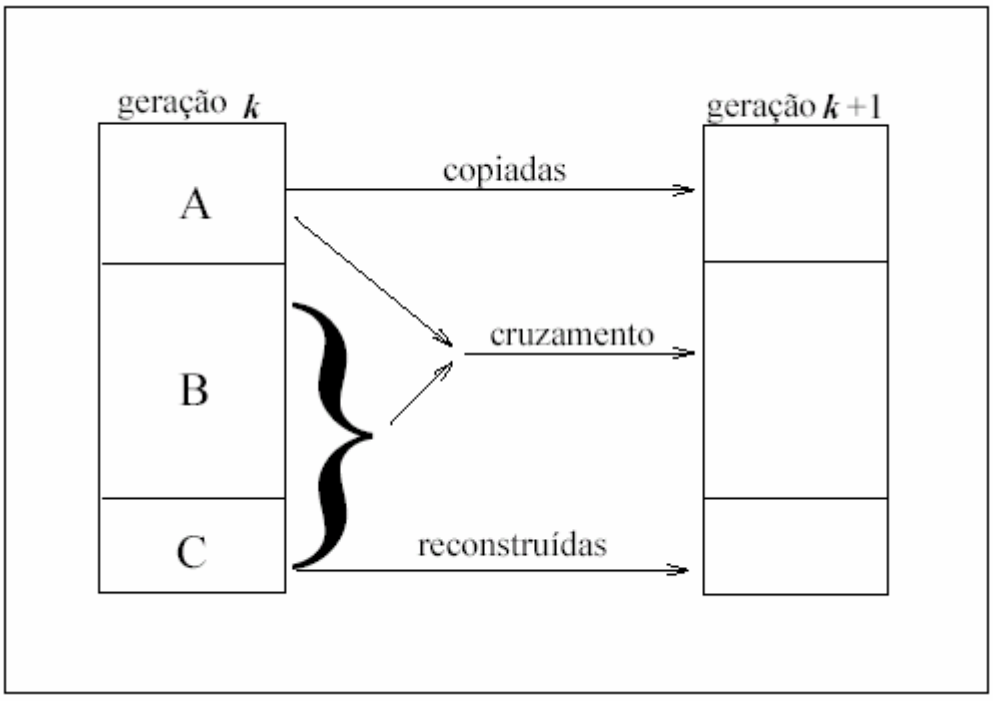

Figura 5 – Evolução da população.

Para gerar a população da geração *k*+1, a partir da geração *k*, os seguintes passos são executados:

- Todos os indivíduos da classe *A* são copiados para a próxima geração.
- Todos os elementos da classe *C* são reconstruídos, ou seja, 5% dos indivíduos de cada geração são construídos utilizando o algoritmo de construção aleatorizado CONSTRUIRINDIVIDUO descrito na Figura 1. É o mesmo procedimento utilizado na criação da população inicial.
- Os indivíduos restantes são provenientes de um cruzamento entre um representante escolhido aleatoriamente na classe *A* com um representante também escolhido aleatoriamente nas classes *B* ou *C* (a operação de cruzamento utilizada será descrita com detalhes na Subseção 2.7).
- Uma busca local é aplicada, com probabilidade de 50%, aos indivíduos provenientes deste cruzamento (o procedimento de busca local utilizado será apresentado na Subseção 2.6). A busca local é aplicada apenas a indivíduos da classe *B*, pois aplicar a indivíduos da classe *A* faria o algoritmo ficar mais lento e, provavelmente, o ganho com isso seria muito pequeno, pois existe uma grande probabilidade de um indivíduo desta classe já ter passado por uma busca local. Aplicar esta busca local na classe *C* vai contra a definição desta classe. A idéia da reconstrução é inserir diversificação na população. Se uma busca local for aplicada, esta diversificação será bastante diminuída.

A estrutura do algoritmo genético implementado neste trabalho é apresentada na Figura 6. Na linha 2, a população inicial, *Pop*, é construída. No laço nas linhas 3-21, esta população é evoluída até atingir um número máximo de iterações, *maxIterações*. Uma população é submetida a uma renovação quando a qualidade da classe elite, classe *A*, não melhora entre trinta gerações consecutivas. Isso é verificado no algoritmo nas linhas 6-10. Como medida de qualidade, usou-se o somatório dos valores objetivos de cada indivíduo pertencente à classe *A*. A estratégia de renovação da população (algoritmo RENOVARPOPULAÇÃO) será discutida com maiores detalhes na Subseção 2.5. Na linha 11, a classe elite *A* da geração *numIterações* é copiada para a geração *numIterações*+1. Na linha 12, a classe *C* é completamente reconstruída para a geração *numIterações*+1. O restante da população da geração *numIterações*+1 é definida nas linhas 13-20 utilizando cruzamentos genéticos entre representantes da classe *A* e representantes das classes *B* ou *C*. Este cruzamento será discutido com maiores detalhes na Subseção 2.7. Na linha 26 é aplicada, com probabilidade de 50%, uma busca local na solução (*filho*) resultante do cruzamento. Assim, é dada uma contribuição para que a população evolua mais depressa. Por fim, na linha 23 é retornada a melhor solução, *s**, encontrada pelo algoritmo.

2.5. RENOVAÇÃO DA POPULAÇÃO

A renovação de uma população é realizada quando algum critério verifica sua estagnação. Neste caso, a população apresenta indivíduos tão parecidos que o cruzamento genético não consegue obter soluções melhores. Assim, é necessário que a população seja renovada para aumentar sua diversidade. No entanto, é importante manter características genéticas adquiridas em gerações anteriores e que são consideradas importantes.

A estratégia neste trabalho foi manter uma parte da classe elite (3% da população) e reconstruir os indivíduos restantes [Ribeiro & Vianna, 2003]. Assim, obtém-se uma diversidade na população e, mantendo os melhores indivíduos, também se mantém um material genético importante para as próximas gerações.

```
Procedimento EVOLUIRPOPULAÇÃO (tamPop, maxIterações)
Entrada
    tamPop - tamanho da população;
    maxIterações - número de iterações a serem realizadas;
Saída
    s* - melhor solução obtida;
Início
01  numIterações ← 0;
02  Pop ← CONSTRUIRPOPULAÇÃO (tamPop);
03  Enquanto numIterações < maxIterações faça
04  Início
05      Ordene a população Pop crescentemente pelo valor objetivo dos indivíduos;
06      Se o critério para renovação da população foi atendido então
07      Início
08          Pop ← RENOVARPOPULAÇÃO (Pop);
09          Ordene a população Pop crescentemente;
10      Fim-se
11      Copie os 25% primeiros indivíduos da população para geração numIterações+1;
12      Construa, com o algoritmo CONSTRUIRINDIVIDUO, 5% da população da geração
            numIterações+1;
13      Para cada indivíduo i da geração numIterações+1 ainda não definido faça
14      Início
15          Escolha aleatoriamente um representante da classe A, pai1;
16          Escolha aleatoriamente um representante da classe B ou C, pai2;
17          filho ← REALIZARCRUZAMENTO (pai1, pai2);
18          Com uma probabilidade de 50%, aplique a busca local em filho;
19          Adicione filho como o i-ésimo indivíduo da geração numIterações+1;
20      Fim-para
21  Fim-enquanto
22  Ordene a população Pop crescentemente pelo valor objetivo dos indivíduos;
23  s* ← o primeiro indivíduo da população pop;
24  Retorne s*;
Fim-EVOLUIRPOPULAÇÃO
```

Figura 6 - Estrutura do algoritmo genético implementado.

Os resultados que serão apresentados na Seção 3 mostram que na maioria das vezes para se obter uma solução de boa qualidade não é necessário que se faça uma renovação da população, ou seja, os resultados obtidos pela evolução da população inicial já são satisfatórios.

2.6. BUSCA LOCAL

Para melhorar as rotas, um algoritmo de busca local é aplicado em algumas soluções. O algoritmo de busca local implementado recebe como entrada a solução *filho* gerada pelo cruzamento e verifica se há arestas desnecessárias na rota, ou seja, seqüências de arestas são analisadas e verifica-se se há alguma outra seqüência de menor custo para substituir o trecho em questão.

2.7. CRUZAMENTO DE INDIVÍDUOS

O operador de cruzamento implementado é uma variação do método ERX [Whitley *et al*., 1991], já utilizado com sucesso em problemas de roteamento [Ochi *et al*., 1998a; Ochi *et al*., 1998b]. O cruzamento é feito escolhendo-se dois pais (podendo ser facilmente estendido para *r* pais), onde um pai pertencente à classe *A* e o outro às classes *B* ou *C*.

Após a escolha dos pais, é feita a separação das arestas que os compõem de acordo com a vizinhança. Por exemplo, sejam as seguintes soluções escolhidas como os pais para o cruzamento:

- Pai 1: (1,5) (5,2) (2,3) (3,7) (7,5) (5,1) (1,4) (4,2) (2,7) (7,2) (2,1) (1,6) (6,8) (8,6) (6,1).
- Pai 2: (1,4) (4,2) (2,5) (5,7) (7,3) (3,2) (2,1) (1,5) (5,2) (2,3) (3,7) (7,2) (2,1) (1,6) (6,8) (8,6) (6,1).

A separação de suas arestas resultaria na estrutura de vizinhança descrita na Tabela 2, onde *Vet* é um vetor que guarda os vizinhos (à direita) diferentes de cada aresta a ser atendida, ou seja, mesmo que uma determinada aresta apareça na rota várias vezes com os mesmos vizinhos, ela só terá o registro de seu vizinho uma única vez.

Tabela 2 – Estrutura de vizinhança.

Vet[1,2]:	*Vet*[4-1]:
Vet[1,4]: (4,2)	*Vet*[4-2]: (2,7) (2,5)
Vet[1,5]: (5,2)	*Vet*[5-1]: (1,4)
Vet[1,6]: (6,8)	*Vet*[5-2]: (2,3)
Vet[2,1]: (1,6) (1,5)	*Vet*[5-7]: (7,3)
Vet[2,3]: (3,7)	*Vet*[6-1]:
Vet[2,4]:	*Vet*[6-8]: (8,6)
Vet[2,5]: (5,7)	*Vet*[7-2]: (2,1)
Vet[2-7]: (7,2)	*Vet*[7-3]: (3,2)
Vet[3-2]: (2,1)	*Vet*[7-5]: (5,1)
Vet[3-7]: (7,5) (7,2)	*Vet*[8-6]: (6,1)

A partir das arestas separadas dos pais de acordo com a vizinhança, a geração da solução *filho* se dá inicialmente com as arestas desmarcadas e escolhe-se uma aresta que parta do nó depósito (v=1). Esta aresta (v,u) é marcada. Como observado no algoritmo CONSTRUIRINDIVIDUOS, a geração do filho também é feita até que todas as arestas se encontrem marcadas. Passo a passo, uma nova aresta (u,w) vizinha de (v,u) é escolhida segundo seu número de vizinhos, ou seja, é escolhido um vizinho da aresta atual que possua a menor quantidade de vizinhos, pois assim as chances de esgotarem os vizinhos de uma determinada aresta diminuem bastante. Esta escolha é feita quando (u,w) possui um número de vizinhos maior que zero.

A aresta (u,w) é inserida na rota, marcada e removida da vizinhança de (v,u) no vetor *Vet*. A aresta (u,w) passa ser a aresta atual e o processo continua.

Se o número de vizinhos de (v,u) for zero, então escolhe-se a aresta (x,z) que possui o caminho mais curto até (v,u). O caminho completo de (v,u) até (x,z) é inserido na rota. A aresta (x,z) é marcada e passa ser a aresta atual.

Como exemplo, sejam as duas soluções pais citadas anteriormente, cuja lista de vizinhança está descrita na Tabela 2. A partir destes pais, será construída uma solução filha, pela estratégia aqui proposta.

Inicialmente, é escolhida, entre as arestas que partem do depósito (v=1), aquela que possui a menor quantidade de vizinhos (para ser escolhida, a aresta deve possuir pelo menos um vizinho). Suponha que a aresta (1,4) seja escolhida. Seu custo e sua demanda são computados. A solução parcial é descrita a seguir.

Solução: (1,4)

Custo: 3

Demanda Total: 3

Como a aresta (1,4) só tem como vizinha a aresta (4,2) – ver Tabela 2 –, esta é escolhida, removida da vizinhança da aresta (1,4) – ver Tabela 3 – e seu custo e sua demanda são computados. A solução parcial é descrita a seguir:

Solução: (1,4) (4,2)

Custo: 12

Demanda Total: 9

Tabela 3 – Estrutura de vizinhança após a inserção da segunda aresta.

Vet[1,2]:	*Vet*[4-1]:
Vet[1,4]:	*Vet*[4-2]: (2,7) (2,5)
Vet[1,5]: (5,2)	*Vet*[5-1]: (1,4)
Vet[1,6]: (6,8)	*Vet*[5-2]: (2,3)
Vet[2,1]: (1,6) (1,5)	*Vet*[5-7]: (7,3)
Vet[2,3]: (3,7)	*Vet*[6-1]:
Vet[2,4]:	*Vet*[6-8]: (8,6)
Vet[2,5]: (5,7)	*Vet*[7-2]: (2,1)
Vet[2-7]: (7,2)	*Vet*[7-3]: (3,2)
Vet[3-2]: (2,1)	*Vet*[7-5]: (5,1)
Vet[3-7]: (7,5) (7,2)	*Vet*[8-6]: (6,1)

A aresta (4,2) tem como vizinhas as arestas (2,7) e (2,5). Dentre elas, é escolhida a aresta que possuir o menor número de arestas vizinhas, mas como as duas possuem a mesma quantidade, é escolhida a que aparecer primeiro na vizinhança da aresta (4,2). A aresta (2,7) é escolhida. Esta é removida da vizinhança da aresta (4,2) – ver Tabela 4 –, seu custo e sua demanda são computados. A solução parcial é descrita a seguir.

Solução: (1,4) (4,2) (2,7)

Custo: 14

Demanda Total: 18

Tabela 4 – Estrutura de vizinhança após a inserção da terceira aresta.

Vet[1,2]:	*Vet*[4-1]:
Vet[1,4]:	*Vet*[4-2]: (2,5)
Vet[1,5]: (5,2)	*Vet*[5-1]: (1,4)
Vet[1,6]: (6,8)	*Vet*[5-2]: (2,3)
Vet[2,1]: (1,6) (1,5)	*Vet*[5-7]: (7,3)
Vet[2,3]: (3,7)	*Vet*[6-1]:
Vet[2,4]:	*Vet*[6-8]: (8,6)
Vet[2,5]: (5,7)	*Vet*[7-2]: (2,1)
Vet[2-7]: (7,2)	*Vet*[7-3]: (3,2)
Vet[3-2]: (2,1)	*Vet*[7-5]: (5,1)
Vet[3-7]: (7,5) (7,2)	*Vet*[8-6]: (6,1)

Seguindo este algoritmo, a solução filha obtida é a seguinte:

Solução: (1,4) (4,2) (2,7) (7,2) (2,1) (1,5) (5,2) (2,3) (3,7) (7,5) (5,1) (1,6) (6,8) (8,6) (6,1)

Custo: 55.

3. RESULTADOS COMPUTACIONAIS

Todos os experimentos computacionais foram realizados em um computador com processador Pentium IV com 3.0GHz de freqüência e com memória principal de 512MBytes. O algoritmo genético híbrido proposto foi implementado em C usando a versão 6.0 do compilador Microsoft Visual C++.

Os experimentos computacionais realizados usaram um conjunto de problemas testes que podem ser acessados no endereço http://www.uv.es/~belengue/carp.html. Este conjunto contem 25 instâncias não-direcionadas construídas por DeArmon (1981), com o número de nós variando entre 7 e 27 e o número de arestas variando entre 11 e 55. As instâncias gdb8 e gdb9 não são usadas porque elas possuem inconsistências [Belenguer & Benavent, 2003].

O algoritmo genético híbrido proposto foi comparado com os limites inferiores apresentados por Belenguer e Benavent (2003). A Tabela 5 apresenta os resultados obtidos. Para cada instância são apresentados nas colunas 2 e 3 o número de nós e o número de arestas, respectivamente. Na coluna 4 é apresentado o limite inferior descrito em [Belenguer & Benavent, 2003]. Nas colunas 5 e 6 são apresentados, respectivamente, o custo e o tempo obtido pelo AG proposto.

Tabela 5 – Resultados comparando o AG proposto com o limite inferior descrito em [Belenguer & Benavent, 2003].

Problema teste	***N***	***|A|***	***Limite inferior***	***AG proposto***	
				Custo	***Tempo(s)***
gdb1	*12*	*22*	*316*	*316*	*0,94*
gdb2	*12*	*26*	*339*	*339*	*4,75*
gdb3	*12*	*22*	*275*	*275*	*0,06*
gdb4	*11*	*19*	*287*	*287*	*0,06*
gdb5	*13*	*26*	*377*	*377*	*15,12*
gdb6	*12*	*22*	*298*	*298*	*0,06*
gdb7	*12*	*22*	*325*	*325*	*0,08*
Gdb10	*27*	*46*	*344*	*358*	*412,52*
Gdb11	*27*	*51*	*303*	*303*	*312,45*
Gdb12	*12*	*25*	*375*	*275*	*2,61*
Gdb13	*22*	*45*	*395*	*395*	*42,03*
Gdb14	*13*	*23*	*450*	*458*	*283,43*
Gdb15	*10*	*28*	*536*	*544*	*99,37*
Gdb16	*7*	*21*	*100*	*100*	*005*
Gdb17	*7*	*21*	*58*	*58*	*0,04*
Gdb18	*8*	*28*	*127*	*127*	*24,00*
Gdb19	*8*	*28*	*91*	*91*	*0,08*
Gdb20	*9*	*36*	*164*	*164*	*2,17*
Gdb21	*8*	*11*	*55*	*55*	*001*
Gdb22	*11*	*22*	*121*	*121*	*0,06*
Gdb23	*11*	*33*	*156*	*156*	*1,19*
Gdb24	*11*	*44*	*200*	*200*	*1,52*
Gdb25	*11*	*55*	*233*	*233*	*471,92*

O algoritmo foi executado dez vezes para cada problema teste. Foram executadas *maxIterações* = 1000 iterações do algoritmo, e contabilizado o tempo em que a melhor solução foi encontrada. Comparando o custo obtido pelo algoritmo proposto com o limite inferior descrito na coluna 4, percebe-se que em 20 dos 23 testes foi obtido um custo igual ao limite inferior. Nos outros 3 testes custos próximos foram obtidos. Isso demonstra a eficiência do algoritmo proposto.

Outra informação importante do algoritmo proposto foi extraída deste experimento: foi verificado que em 67% das execuções não foi necessário realizar o procedimento de renovação da população (descrito na Subseção 2.5) para obter a melhor solução, e em 17% este procedimento não foi realizado mais de duas vezes.

4. CONCLUSÃO

O problema de roteamento em arcos capacitados é um problema NP-difícil que possui diversas aplicações práticas, tais como a coleta de lixo urbano e a inspeção de linhas de força.

O algoritmo genético híbrido proposto neste trabalho se mostrou eficiente obtendo, entre as 23 instâncias testadas, 20 soluções de custo igual ao limite inferior descrito em [Belenguer & Benavent, 2003]. Nas outras instâncias, resultados próximos foram obtidos.

Também foi verificado que em 67% das execuções não foi necessário realizar o procedimento de renovação da população para obter a melhor solução, e em 17% este procedimento não foi realizado mais de duas vezes. Esses resultados mostram que o algoritmo proposto é bastante eficiente, pois na maioria das execuções este atingiu a melhor solução (na grande maioria dos casos, o limite inferior) sem precisar realizar a renovação da população.

AGRADECIMENTOS

Este trabalho foi parcialmente financiado pela Prefeitura Municipal da cidade de Campos dos Goytacazes.

REFERÊNCIAS

BEAN, J. C.. Genetic algorithms and random keys for sequencing and optimization. **ORSA Journal on Computing**, v. 6, 154-160, 1994.

BELENGUER, J.M.; BENAVENT, E. A cutting plane algorithm for the capacitated arc routing problem. **Computers and Operations Research**, v. 30, n. 15, 705-728, 2003.

BURIOL, L. S.; RESENDE, M. G. C.; RIBEIRO, C. C.; THORUP, M. (2003). A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. *Relatório técnico*.

CORBERÁN, A.; MARTÍ, R.; ROMERO, A. (2000). Heuristics for the mixed rural postman problem. **Computers and Operations Research**, v. 27, 183-203.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos.** Rio de Janeiro: Editora Campus, 2ª edição. 2002.

DEARMON, J. S. . **A Comparison of heuristics for the capacitated chinese postman problem**. Master's Thesis, The University of Maryland at College Park, MD, USA, 1981.

DRUMMOND, L. M. A.; OCHI, L. S.; VIANNA, D. S. . An asynchronous parallel metaheuristic for the period vehicle routing problem. **Future Generation Computer Systems**, 17:379-386, 2001.

EGLESE, R. W. .Routing winter gritting vehicles. **Discrete Applied Mathematics**, v. 48, n. 3, 231-244, 1994.

EGLESE, R. W.; LI, L. Y. O. . A tabu search based heuristic for arc routing with a capacity constraint and time deadline. Em: I.H. Osman e J.P. Kelly (editores), **Metaheuristics: theory and applications**, New York: Kluwer, 633-650, 1996.

GLOVER, F. . Tabu search for nonlinear and parametric optimization with links to genetic algorithms. **Discrete Applied Mathematics**, v. 49, 231-255, 1994.

GLOVER, F. . Scatter search and star-paths: Beyond the genetic metaphor. **OR Spektrum**, v. 17:125-137, 1995.

GOLDEN, B. L.; WONG, R. T. . Capacitated arc routing problems. **Networks**, v. 11, 305-315, 1981.

GOLDEN, B. L.; DEARMON, J. S.; BAKER, E. K. . Computational experiments with algorithms for a class of routing problems. **Computers and Operation Research**, v. 10, n. 1, 47-59, 1983.

HERTZ, A.; LAPORTE, G.; NANCHEN-HUGO, P. . Improvement procedures for the undirected rural postman problem. **INFORMS Journal on Computing**, v. 11, n. 1, 53-62, 1999.

HERTZ, A.; LAPORTE, G.; MITTAZ, M. . A tabu search heuristic for the capacitated arc routing problem. **Operations Research**, v. 48, n. 1, 129-135, 2000.

HIRABAYASHI, R.; SARUWATARI, Y.; NISHIDA, N. . Tour construction algorithm for the capacitated arc routing problem. **Asia-Pacific Journal of Operational Research**, v. 9, 155-175, 1992.

HOLLAND, J. . **Adaptation in natural and artificial systems**. MIT Press, 2ª edição. 1992.

MICHALEWICZ, Z. . **Genetic algorithm + data structures = evolution programs**. Springer-Verlag, 3ª edição, 1996.

OCHI, L. S.; DRUMMOND, L. M. A.; VIANNA, D. S. . An evolutionary hybrid metaheuristic for solving the vehicle routing problem with heterogeneous fleet. ***Lecture Notes in Computer Science***, n. 1391:187-195, 1998.

OCHI, L. S.; DRUMMOND, L. M. A.; VIANNA, D. S.; VICTOR, A. O. . A parallel evolutionary algorithms for solving the vehicle routing problem with heterogeneous fleet. **Future Generation Computer Systems**, v. 14, 285-292, 1998.

PEARN, W. L. . Augment-insert algorithms for the capacitated arc routing problem, **Computers and Operations Research**, v. 18, n. 2, 189-198, 1991.

RIBEIRO, C. C.; VIANNA, D. S. . A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path-relinking, **Revista Tecnologia da Informação - RTInfo**, v.3, p.67-70, 2003.

ROCHAT, Y.; TAILLARD, E. . Probabilistic diversification and intensification on local search for vehicle routing. **Journal of Heuristics**, v. 1, 147-167, 1995.

SCHWEFEL, H. P. . **On the evolution of evolutionary computation**. In: Zurada, J. M.; Marks, R. ; Robinson, C. J., editores, *Computation Intelligence - Imitating Life*, p. 116-124. IEEE Press, 1994.

VIANNA, D. S.; OCHI, L. S.; DRUMMOND, L. M. A. . A parallel hybrid evolutionary metaheuristic for the period vehicle routing problem with heterogeneous fleet. **Lecture Notes in Computer Science**, n. 1388, 216-225, 1999.

WHITLEY, D.; STARKWEATHER, T.; SHANER, D. . **The traveling salesman, sequence scheduling: quality solutions using genetic edge recombination. Handbook of Genetic Algorithms**. New York: Van Nostrand Reinhold, 1991.

An evolutionary algorithm for the capacitated arc routing problem

Dalessandro Soares Vianna[1], dalessandro@ucam-campos.br.

Roberta Claudino Barreto Pessanha Gomes[1], roberta.claudino@gmail.com.

[1] Universidade Candido Mendes – UCAM-Campos
Núcleo de Pesquisa e Desenvolvimento em Informática - NPDI,
Campos dos Goytazazes, RJ 28040-320, Brasil.

*Received: March, 2006 / Accepted: May, 2006

ABSTRACT

The Capacitated Arc Routing Problem (CARP) consists of visiting a subset of edges of the graph that describes the problem. CARP applications include urban waste collection and inspection of power lines. The CARP is NP-hard, even in the single-vehicle case (called Rural Postman Problem). In this case, the use of metaheuristics is an efficient solution strategy. This paper proposes a hybrid genetic algorithm for the CARP, which is tested on available instances of the literature. The results obtained until now show the effectiveness of the proposed algorithm when it is compared with lower bounds described in the literature.

Keywords: Capacitated arc. Routing problem. Vehicle routing. Genetic algorithms.