

## 5

### Hibridização dos Métodos

#### 5.1

##### Introdução

Conforme comprovado pelos resultados obtidos pelo algoritmo *Branch-Cut-and-Price* apresentado no capítulo 3, abordagens exatas nem sempre são capazes de encontrar soluções inteiras de boa qualidade em um tempo computacional razoável. Nesse contexto, tem-se abordagens heurísticas, as quais procuram encontrar soluções de boa qualidade sem o requisito de encontrar soluções ótimas, nem muito menos de provar a otimalidade. Muitos desses algoritmos conseguem gerar soluções de qualidade em um baixo tempo computacional. Tal atrativo aliado ao fato de que desenvolver tais algoritmos não exige muitos conhecimentos prévios no que diz respeito a técnicas de programação matemática, tem tornado essas técnicas muito aplicadas em problemas reais.

Apesar dessas duas áreas de pesquisa terem evoluído praticamente de forma independente, nos últimos anos, surgiram alguns trabalhos que tentam mesclar as duas vertentes.

Este capítulo primeiramente mostra uma classificação de abordagens que combinam técnicas exatas e heurísticas (Seção 5.2). Em seguida, destaca o enfoque deste trabalho no uso do algoritmo *Branch-Cut-and-Price (BCP)* como uma busca local através da adição de cortes (Seção 5.3). Essa busca local pode ser utilizada no contexto de diferentes metaheurísticas.

Posteriormente, é descrita uma heurística utilizada para gerar soluções inteiras a partir de soluções fracionárias (Seção 5.4). Com essa heurística, mesmo sem executar a busca exata até alcançar a otimalidade, é possível encontrar soluções inteiras durante o processo, o que reduz o tempo total gasto pela metaheurística para convergir.

Em seguida, é apresentado o Algoritmo Evolutivo que utiliza como operador de *crossover* uma busca elipsoidal (Seção 5.5). Finalmente, os resultados do algoritmo são analisados (Seção 5.6) e são apresentadas as conclusões extraídas dos experimentos realizados com essa abordagem (Seção 5.7).

## 5.2

### Combinando técnicas exatas e heurísticas

As abordagens que combinam técnicas exatas e heurísticas foram classificadas por Puchinger e Raidl (PR05) em duas categorias:

- Combinações Colaborativas: essa categoria engloba abordagens nas quais algoritmos exatos e heurísticos trocam informações. Entretanto, os dois algoritmos são independentes, ou seja, nenhum deles é parte do outro. Eles podem executar sequencialmente, intercaladamente ou em paralelo;
- Combinações Integrativas: essa categoria abrange abordagens nas quais há um algoritmo mestre e um ou mais algoritmos subordinados ou escravos. O mestre pode ser um algoritmo exato e os escravos meta-heurísticas ou vice-versa.

#### 5.2.1

##### Combinações Colaborativas

Há diversos trabalhos na literatura que fazem uso de Combinações Colaborativas. Como exemplo de combinação colaborativa sequencial pode-se citar o trabalho de Vasquez e Hao (VH01). Eles resolvem heurísticamente o Problema da Mochila Multi-Restrito reduzindo e particionando o espaço de busca através de restrições que fixam o número total de itens a serem selecionados. Resolvendo-se uma relaxação do problema original, obtém-se o limite a ser utilizado nessas restrições. Uma busca tabu paralela é realizada em cada parte do espaço de busca a partir de uma solução inicial obtida da relaxação linear do problema parcial.

Há também casos de trabalhos que utilizam Combinações Colaborativas de maneira Intercalada. Nessa linha de pesquisa, pode-se referenciar o trabalho de Gallardo et al. (GCF05) que hibridizaram um algoritmo evolutivo e um *branch-and-bound* aplicado ao Problema da Mochila Multidimensional. Os algoritmos são executados de maneira intercalada e cooperam trocando informações entre eles. O Algoritmo Evolutivo fornece limites para o *branch-and-bound* enquanto que o *branch-and-bound* fornece melhores soluções parciais para o algoritmo evolutivo.

#### 5.2.2

##### Combinações Integrativas

As combinações integrativas têm sido estudadas na literatura e aplicadas principalmente nos seguintes casos:

- **Resolver exatamente problemas relaxados:** consiste em resolver o problema relaxado e tirar proveito das informações da solução relaxada para gerar soluções de qualidade para o problema original. Pode-se referenciar o trabalho de Chu e Beasley (CB98) que utilizam a solução do problema relaxado e do seu dual no contexto de um algoritmo genético híbrido para o Problema da Mochila Multi-Restrito;
- **Realizar busca em grandes vizinhanças:** se as vizinhanças forem escolhidas adequadamente, elas podem vir a ser muito grandes. Nesse sentido, faz-se necessário utilizar buscas eficientes para grandes vizinhanças. Essa técnica é conhecida como *Very Large-Scale Neighborhood Search*. Burke et al. (BCK01) apresentaram uma *Variable Neighborhood Search* para o Problema do Caixeiro Viajante Assimétrico que incorporou um algoritmo exato na parte da busca local, denominado *HyperOpt* que explora exaustivamente regiões promissoras e relativamente grandes do espaço de busca do problema.

### 5.3

#### Combinando Metaheurísticas com BCP

Como já foi visto, há uma dificuldade de se encontrar soluções inteiras para o TOP usando o algoritmo BCP. Partindo dessa premissa e devido ao grande sucesso de abordagens que mesclam idéias de heurísticas com resolvidores MIP, surgiu a idéia de propor, neste trabalho, cortes a serem adicionados à formulação utilizada no BCP. Com esse cortes, foi possível, a partir de uma solução inteira qualquer conhecida, realizar buscas em diferentes vizinhanças por soluções melhores.

Até o momento, não se tem conhecimento de outros trabalhos que tenham utilizado um algoritmo *Branch-Cut-and-Price* como busca local através da adição de restrições que limitem o espaço de busca a uma vizinhança em torno de uma ou mais soluções incumbentes.

#### 5.3.1

##### Busca Circular

Inicialmente foi proposta uma busca circular em torno de uma solução composta por dois vetores de variáveis 0-1:  $\bar{x}$ ,  $\bar{y}$ , onde:

- $\bar{x}$  é conjunto de valores das variáveis  $x_a$  que indicam se um determinado arco  $a \in A$  é utilizado
- $\bar{y}$  é conjunto de valores das variáveis  $y_v$  que indicam se um determinado vértice  $v \in V$  é visitado.

Sabe-se que o número de arcos utilizados em uma solução pode ser calculado da seguinte forma:

$$n_a = \sum_{v \in V | \bar{y}_v = 1} y_v + m \quad (5-1)$$

É fácil chegar à equação 5-1, pois, de cada vértice visitado parte um arco e há mais  $m$  arcos que partem do ponto inicial.

Uma busca circular consiste em adicionar restrições de modo a reduzir o espaço de busca a uma vizinhança de tamanho  $k$  em relação à solução base  $\bar{x}$ . São restrições semelhantes às utilizadas no *local branching*, entretanto, neste trabalho, a restrição é aplicada numa formulação de BCP.

Para adicionar o corte à formulação, as variáveis  $x_a$  são traduzidas em termos de variáveis  $\lambda_j$  que representam as rotas não elementares, conforme a equação 5-2.

$$\sum_{l=0}^L \sum_{j \in Q} g_a^{lj} \cdot \lambda_j = x_a \quad \forall a \in A \quad (5-2)$$

A restrição que representa a vizinhança  $k$ -*Opt* é apresentada em 5-3

$$\sum_{v \in V | y_v = 1} y_v + \sum_{v \in V | y_v = 0} (1 - y_v) + \sum_{a \in A | x_a = 1} x_a + \sum_{a \in A | x_a = 0} (1 - x_a) \geq n_t - k \quad (5-3)$$

onde:

- $n_t$  é o número de pontos visitados por  $\bar{x}$  somado com o número de arcos utilizados por  $\bar{x}$  conforme mostra a equação 5-4
- $k$  é o raio da busca local.

$$n_t = \sum_{v \in V | y_v = 1} y_v + n_a = 2 \left( \sum_{v \in V | y_v = 1} y_v \right) + m \quad (5-4)$$

Por essa restrição ser muito densa, neste trabalho, ela foi simplificada desconsiderando-se as variáveis que possuem valor 0 (zero) na solução base  $\bar{x}$ .

Assim, tem-se em 5-5, a restrição simplificada da vizinhança circular:

$$\sum_{v \in V | y_v = 1} y_v + \sum_{a \in A | x_a = 1} x_a \geq n_t - k \quad (5-5)$$

O pseudocódigo do algoritmo da busca circular é apresentado no Algoritmo 5.

---

**Algorithm 5:** Busca Circular

---

**Input:**  $K_{min}$  raio de busca mínimo

**Input:**  $K_{max}$  raio de busca máximo

**Input:**  $step$  indica de quanto irá aumentar o valor de  $k$  a cada aumento de tamanho de vizinhança

**Input:**  $\bar{x}$  solução base

**Input:**  $x_{best}$  melhor solução encontrada

```

1  $k \leftarrow k_{min}$ ;
2 while  $k < K_{max}$  do
3   adicionar ao BCP restrição  $\Delta(\bar{x}, x) \leq k$ ;
4    $x_{best} \leftarrow \text{executaRBCP}()$ ;
5   if  $x_{best}$  é melhor que  $\bar{x}$  ou é ótimo local then
6      $\text{return } x_{best}$ ;
7   else
8     aumenta a vizinhança;
9      $k \leftarrow k + step$ ;

```

---

O algoritmo recebe como entrada um raio inicial que determina o tamanho da vizinhança. Se a busca não obtiver uma solução de melhor qualidade do que a solução base  $\bar{x}$ , o algoritmo, a cada iteração incrementa o tamanho da vizinhança de um valor  $step$  pré-definido.

Pode-se perceber que a vizinhança  $k\text{-Opt}$  representa o conjunto de soluções que estão a uma distância  $k$  da solução  $\bar{x}$ . Visualmente pode-se considerar  $\bar{x}$  o centro de um círculo de raio  $k$  e, os pontos desse círculo representam as soluções vizinhas de  $\bar{x}$ .

### 5.3.2

#### Busca Elipsoidal

Pigatti et al. (PPU04) propuseram uma vizinhança semelhante à busca circular, porém centrada em duas soluções base  $s_1$  e  $s_2$ . A analogia com elipse se dá, pois uma elipse possui dois pontos fixos  $F_1$  e  $F_2$ , denominados focos da elipse. Na vizinhança elipsoidal tais pontos fixos simbolizam as duas soluções base  $s_1$  e  $s_2$ . Uma elipse é definida pelo conjunto de todos os pontos do plano, cuja soma das distâncias  $r_1$  e  $r_2$  para os dois pontos fixos respectivamente é um valor constante e positivo  $a$ . Considerando-se que as soluções são pontos multi-dimensionais, essa área da elipse representa o conjunto de soluções vizinhas de  $s_1$  e de  $s_2$  simultaneamente.

Seja  $Y_1$  o conjunto de variáveis  $y_v$  com valor igual a 1 na solução  $s_1$ . O conjunto  $X_1$  contém todos os arcos com valor igual a 1 em  $s_1$ . De maneira análoga,  $Y_2$  o conjunto de variáveis  $y_v$  com valor igual a 1 na solução  $s_2$  e  $X_2$  contém todos os arcos com valor igual a 1 em  $s_2$ .

Na vizinhança  $K$ -Opt, o valor  $n_t$  representava o número de vértices visitados somado com o número de arcos utilizados pela solução base  $\bar{x}$ . De modo semelhante, no caso da vizinhança elipsoidal é necessário calcular esse valor para cada uma das soluções base. Assim, tem-se que:

$$n_t^1 = 2 \sum_{v \in V | y_v^1 = 1} y_v + m \quad (5-6)$$

$$n_t^2 = 2 \sum_{v \in V | y_v^2 = 1} y_v + m \quad (5-7)$$

Como é necessário apenas um valor de  $n_t$  para ser adicionado no lado direito da restrição do corte, será utilizado o valor máximo entre os valores  $n_t$  das duas soluções base.

$$n_t^{max} = \max(n_t^1, n_t^2)$$

Portanto, o corte elipsoidal pode ser representado pela restrição seguinte, onde  $k$  representa o tamanho da vizinhança elipsoidal:

$$\begin{aligned} & \sum_{v \in V | y_v^1 = 1, y_v^2 = 0} y_v + \sum_{v \in V | y_v^1 = 0, y_v^2 = 1} y_v + 2 \sum_{v \in V | y_v^1 = 1, y_v^2 = 1} y_v + \\ & \sum_{a \in A | x_a^1 = 1, x_a^2 = 0} x_a + \sum_{a \in A | x_a^1 = 0, x_a^2 = 1} x_a + 2 \sum_{a \in A | x_a^1 = 1, x_a^2 = 1} x_a \\ & \geq n_t^{max} + |Y_1 \cap Y_2| + |X_1 \cap X_2| - k \end{aligned} \quad (5-8)$$

Como  $\sum_{v \in V | y_v^1 = 1, y_v^2 = 1} y_v$  é igual a  $|Y_1 \cap Y_2|$  e  $\sum_{a \in A | x_a^1 = 1, x_a^2 = 1} x_a$  é igual a  $|X_1 \cap X_2|$  pode-se considerar que as restrições 5-8 e 5-9 são equivalentes:

$$\begin{aligned} & \sum_{v \in V | y_v^1 = 1, y_v^2 = 0} y_v + \sum_{v \in V | y_v^1 = 0, y_v^2 = 1} y_v + \sum_{v \in V | y_v^1 = 1, y_v^2 = 1} y_v \\ & \sum_{a \in A | x_a^1 = 1, x_a^2 = 0} x_a + \sum_{a \in A | x_a^1 = 0, x_a^2 = 1} x_a + \sum_{a \in A | x_a^1 = 1, x_a^2 = 1} x_a \\ & \geq n_t^{max} - k \end{aligned} \quad (5-9)$$

O pseudocódigo do algoritmo da busca elipsoidal é apresentado no

Algoritmo 6.

---

**Algorithm 6:** Busca Elipsoidal

---

**Input:**  $K_{min}$  raio de busca mínimo

**Input:**  $K_{max}$  raio de busca máximo

**Input:**  $step$  indica de quanto irá aumentar o valor de  $k$  a cada aumento de tamanho de vizinhança

**Input:**  $\bar{s}_1$  primeira solução base

**Input:**  $\bar{s}_2$  segunda solução base

**Input:**  $x_{best}$  melhor solução encontrada

```

1  $k \leftarrow k_{min};$ 
2 while  $k < K_{max}$  do
3   adicionar ao BCP restrição  $\Delta((\bar{s}_1), (\bar{s}_2), x) \leq k;$ 
4    $x_{best} \leftarrow \text{executaRBCP}();$ 
5   if  $x_{best}$  é melhor que  $\bar{s}_1$  ou que  $\bar{s}_2$  ou é ótimo local then
6      $\text{return } x_{best};$ 
7   else
8     aumenta a vizinhança;
9      $k \leftarrow k + step;$ 

```

---

### Introduzindo arcos extras

Em geral, as rotas devem atender ao máximo de clientes possível, respeitando a duração máxima das rotas  $L$ . Para isso, é imprescindível que as sequências das rotas mantenham-se otimizadas ao longo do processo de resolução.

Devido à importância da característica de *Problemas de Caixeiro Viajante* inerente ao TOP, este trabalho propõe uma maneira de permitir ao resolvidor realizar trocas na sequência de visitação dos clientes sem ter para isso que usar a folga  $K$  da busca elipsoidal.

Seja  $T$ , um conjunto de arcos extras que podem substituir outros arcos da solução sem consumir nenhuma unidade do parâmetro  $k$ .

$$\sum_{v \in V | y_v^1 = 1 y_v^2 = 0} y_v + \sum_{v \in V | y_v^1 = 0 y_v^2 = 1} y_v + \sum_{v \in V | y_v^1 = 1 y_v^2 = 1} y_v \quad (5-10)$$

$$\sum_{a \in A | x_a^1 = 1 x_a^2 = 0} x_a + \sum_{a \in A | x_a^1 = 0 x_a^2 = 1} x_a + \sum_{a \in A | x_a^1 = 1 x_a^2 = 1} x_a \quad (5-11)$$

$$+ \sum_{a \in T} x_a \geq n_t^{max} - k \quad (5-12)$$

A Figura 5.1 apresenta uma solução inteira qualquer para o TOP. O conjunto de arcos extras  $T$  é definido como sendo a união de três conjuntos de arcos a saber:

- O conjunto de arcos que partem do ponto inicial e chegam em qualquer vértice visitado pela solução original, conforme ilustra a Figura 5.2;
- O conjunto de arcos que chegam ao ponto final advindos de qualquer vértice visitado pela solução original, conforme ilustra a Figura 5.3;
- Os arcos do *giant-tour*, ou seja, os arcos que conectam um vértice  $i$  a um vértice  $k$ , de modo que, na solução original, exista um e somente um vértice  $j$  que é visitado entre  $i$  e  $k$ . Considera-se todas as rotas como sendo um único *tour*, como se todas as visitas fossem realizadas por um só veículo.

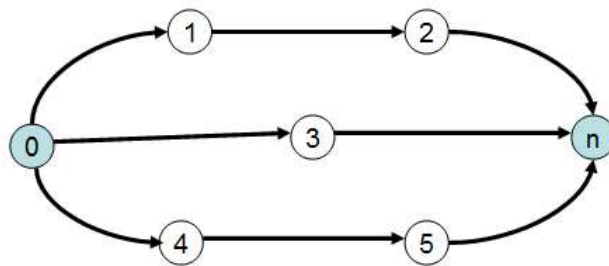


Figura 5.1: Arcos Originais

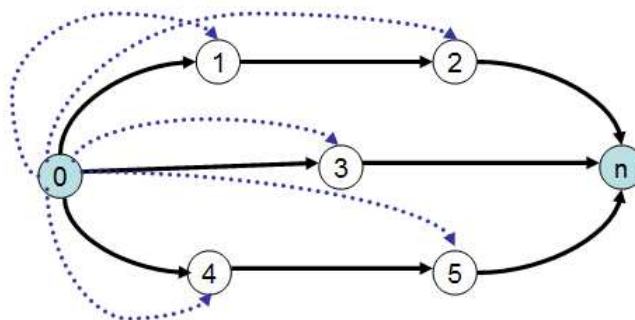


Figura 5.2: Arcos do ponto inicial



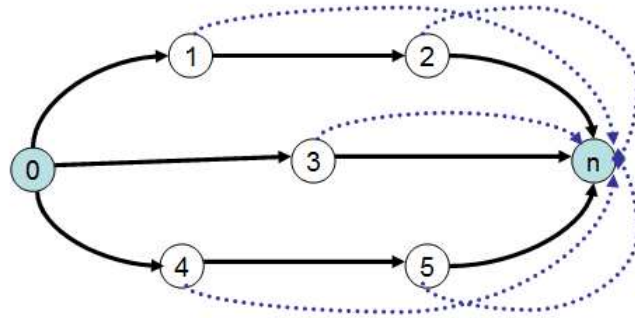


Figura 5.3: Arcos para o ponto final

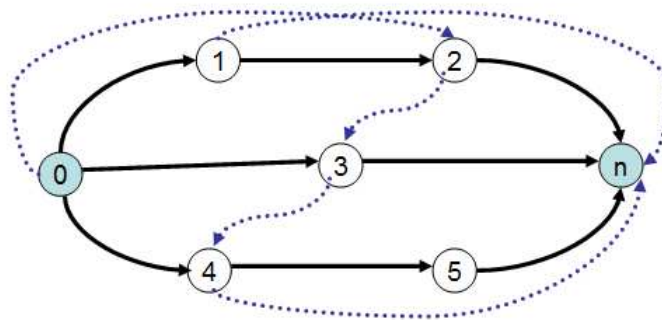


Figura 5.4: Arcos do Giant-tour

#### 5.4 Heurística baseada na Solução Linear

Apesar do corte elipsoidal reduzir o espaço de busca do problema, o tempo computacional da busca é bastante elevado até que ela encontre uma solução inteira. A maior parcela desse tempo é gasta no algoritmo de geração de colunas.

Como são necessárias várias iterações da busca elipsoidal para se obter uma solução de boa qualidade, a questão do tempo computacional torna-se ainda mais preocupante.

Para contornar esse problema, neste trabalho é proposta a idéia de não se executar a busca elipsoidal até que seja encontrada uma solução inteira pelo algoritmo BCP. A idéia consiste em executar o BCP até um determinado nível da árvore e, não se obtendo solução inteira, aplicar uma heurística que utilize informações da solução da relaxação linear dos nós da árvore do BCP. Como o corte elipsoidal faz parte da formulação de todos os nós da árvore, a solução da relaxação linear também traz consigo características inerentes às duas soluções base. Desse modo, a solução resultante da heurística aplicada

sobre o resultado da relaxação linear irá gerar soluções inteiras que herdam vértices e arcos presentes nas soluções base.

Por essa razão, uma busca elipsoidal traz consigo o mesmo conceito de herança de características de indivíduos ancestrais presente nos operadores de *crossover* dos algoritmos evolutivos.

Dessa maneira, uma iteração da busca elipsoidal torna-se muito mais rápida e uma solução inteira sempre é obtida. Neste trabalho, foi proposta uma heurística simples, de baixo custo computacional e que encontra soluções razoáveis.

O pseudocódigo dessa heurística executada nos nós da árvore do algoritmo BCP é apresentado no Algoritmo 7:

---

**Algorithm 7:** Heurística de Arredondamento

---

**Input:**  $\Lambda$  conjunto variáveis de rotas com valores positivos

**Input:**  $m$  o número de veículos da frota

**Output:** Solução inteira  $s$

```

1 Ordena o conjunto  $\Lambda$  em ordem decrescente de valor da solução
  fracionária;
2  $s \leftarrow \text{CreateNewSolution}()$ ;
3  $vehicle \leftarrow 0$ ;
4 foreach  $\lambda_j \in \Lambda$  do
5   if  $vehicle = m$  then
6      $\lfloor$  break;
7   foreach  $v \in \lambda_j$  do
8     if  $v \notin s$  then
9        $\lfloor$  insertVertice( $v, s, vehicle$ );
10     $vehicle++$ ;
11 tenta inserir outros vértices em  $s$ ;
12 return  $s$ ;
```

---

Essa heurística recebe uma solução fracionária resultante da relaxação linear de um nó qualquer da árvore do BCP. Primeiramente as rotas são ordenadas em ordem decrescente do valor da variável da respectiva rota na relaxação linear. Então, tenta-se adicionar as rotas completas à nova solução que está sendo gerada. Entretanto, para garantir a viabilidade da nova solução, não se permite a inserção de vértices repetidos provenientes das rotas não elementares da relaxação linear. Após testar a inserção de todos os vértices de uma rota, incrementa-se o contador de veículos para dar início à geração da rota seguinte. Esse processo se repete até que as  $m$  rotas tenham sido geradas.

## 5.5

### Algoritmo Evolutivo

Este trabalho se propõe a utilizar as buscas circulares e elipsoidais como poderosas ferramentas capazes de explorar grandes vizinhanças. Essas buscas locais podem ser incorporadas ao contexto de diferentes metaheurísticas como VNS, Algoritmos Genéticos, Busca Tabu, dentre outras.

Devido à grande semelhança da busca elipsoidal com os operadores de *crossover* dos Algoritmos Evolutivos, aliado ao crescente uso de buscas locais nesse tipo de algoritmos, os denominados Algoritmos Meméticos, optou-se, neste trabalho, por testar a busca primeiramente em um Algoritmo Evolutivo.

O pseudocódigo abaixo descreve o esquema geral do Algoritmo Evolutivo desenvolvido que utiliza a busca elipsoidal como operador de *crossover*.

#### 5.5.1

##### Representação dos Cromossomos

Um cromossomo do Algoritmo Evolutivo proposto é equivalente a uma solução para o problema TOP. A representação do cromossomo é binária, pois consiste em um vetor de variáveis do tipo 0-1. Fazem parte do cromossomo tanto as variáveis  $y_v$  que indicam os vértices que são visitados pela solução, como as variáveis  $x_a^l$  que indicam quais arcos são utilizados e em que instantes acontecem os seus percursos.

#### 5.5.2

##### Função de avaliação

A função de avaliação dos cromossomos é responsável por mensurar a aptidão dos cromossomos. Essa função é equivalente à função objetivo das formulações, ou seja, visa a maximizar o prêmio total coletado e é definida em 5-13

$$\sum_{v \in V^-} p_v \cdot y_v \quad (5-13)$$

onde  $p_v$  é o prêmio associado ao vértice  $v$ .

#### 5.5.3

## Esquema geral do Algoritmo Evolutivo

O esquema geral do Algoritmo Evolutivo é apresentado no seguinte pseudocódigo 8:

---

### Algorithm 8: Algoritmo Evolutivo com Crossover Elipsoidal

---

**Input:**  $n$  vértices

**Input:**  $m$  veículos

**Input:**  $L$  duração máxima das rotas

**Input:**  $MaxIter$  número máximo de iterações do Algoritmo Evolutivo

**Output:** Best solution  $s_{best}$

```

1 TOPSolution iniSol = GreedyHeuristic( $n, m, L$ );
2 TOPSolution[] population = RandomShakeHeuristic(iniSol);
3 TOPSolution  $s_{best}$  = bestSol(population);
4 int iter = 0;
5 while iter <  $MaxIter$  do
6     TOPSolution parent1 = randomSelection(population);
7     TOPSolution parent2 = randomSelection(population);
8     TOPSolution newSol = ellipsoidalRBCP(parent1, parent2);
9     updatePop(population, newSol,  $s_{best}$ );
10    iter++;

```

---

Inicialmente é gerada uma solução inicial e, a partir dessa solução é gerada uma população inicial através de uma heurística. Em seguida iniciam-se as iterações que consistem em selecionar pares de cromossomos e realizar buscas elipsoidais em torno dessas soluções. Essas buscas são realizadas em grandes vizinhanças e o algoritmo converge em relativamente poucas, porém lentas iterações em comparação com algoritmos evolutivos tradicionais.

### 5.5.4

#### População Inicial

A função *GreedyHeuristic* é uma heurística gulosa utilizada apenas para gerar uma solução inicial. Essa heurística insere, a priori, os vértices que possuem os maiores prêmios, respeitando a restrição do tempo limite de duração das rotas.

Outra forma de gerar solução inicial que foi testada neste trabalho foi executar somente o nó raiz do algoritmo BCP e aplicar sobre a solução da relaxação linear a mesma heurística utilizada na busca elipsoidal para encontrar soluções inteiras.

A função *RandomShakeHeuristic* gera um conjunto de soluções baseadas na solução inicial. Esse algoritmo cria subconjuntos aleatórios (com 3 vértices

no máximo) que são candidatas a serem removidos da solução. De maneira análoga são criados subconjuntos de vértices candidatas a serem adicionados à solução. A seleção dos vértices que poderão fazer parte dos conjuntos a serem removidos é probabilística e depende do prêmio a ele associado. Esse algoritmo realiza operações de troca entre subconjuntos de vértices visitados por subconjuntos de vértices não visitados na solução original, garantindo a viabilidade das novas soluções geradas. O conjunto de soluções geradas por este algoritmo é utilizado como população inicial no contexto do Algoritmo Evolutivo.

### 5.5.5

#### Crossover

Os operadores de *crossover* tradicionais recebem dois cromossomos selecionados pelo método da roleta ou pelo método do torneio ou ainda por algum outro método de seleção de natureza aleatória e gera novos cromossomos fazendo uso de parte dos cromossomos ancestrais ou de alguma informação relacionada à ordem ou frequência de bits dos cromossomos originais.

Todavia, devido à restrição do tempo máximo de duração das rotas  $L$ , a aplicação pura desses métodos nem sempre resultaria em soluções viáveis. Além disso, por serem operadores de natureza genérica, tais operadores não consideram as características do problema, ficando a cargo da aleatoriedade a chance de se obter boas soluções.

Nesse trabalho, optou-se por experimentar uma nova abordagem para o operador de *crossover*. A idéia foi utilizar a busca elipsoidal em torno de dois cromossomos ancestrais para gerar um novo cromossomo. Esse operador de *crossover* difere dos tradicionais nos seguintes pontos:

- É um operador específico para o problema. Embora a idéia de usar cortes elipsoidais em um algoritmo *BCP* seja aplicável a uma gama de problemas de otimização combinatória, cada operador em si é específico por estar sendo executado sobre um *BCP* específico para o problema. Desse modo, intuitivamente, utilizar o conhecimento do problema deve trazer bons resultados;
- Trata-se de um operador otimizado. Ao contrário dos operadores tradicionais que visam apenas a gerar soluções viáveis utilizando características dos cromossomos ancestrais, neste operador realiza-se uma busca por um cromossomo filho com a máxima qualidade possível. O tempo de duração dessa busca é parametrizável, de modo que, dependendo da aplicação, pode-se parar a busca a qualquer instante e ter como

novo cromossomo, a melhor solução encontrada na vizinhança até o momento;

- A busca elipsoidal já assegura a viabilidade da nova solução gerada, evitando assim a necessidade de se criar um mecanismo à parte para garantir que o novo cromossomo pertença ao domínio do problema.

## 5.6

### Resultados Obtidos

A Tabela 5.1 a seguir apresenta uma prévia dos resultados obtidos com o Algoritmo Evolutivo que tem como operador de *crossover* uma busca elipsoidal em torno de duas soluções base. A Tabela 5.1 tem os seguintes campos:

- *Inst* : nome da instância;
- *n* : número de vértices;
- *m* : número de veículos da frota;
- *L* : duração máxima das rotas;
- *IniSol* : solução inicial;
- *Pop* : tamanho da população do Algoritmo Evolutivo;
- *OurSol* : melhor solução obtida neste trabalho;
- *Time* : tempo computacional total referente à abordagem deste trabalho;
- *ASH* : melhores resultados do trabalho Archetti et al. (AHS07) de Archetti, Speranza and Hertz.

Tabela 5.1: Tamanho da Vizinhança  $k$  : 14 a 18

Inst	n	m	L	IniSol	Pop	OurSol	Time	ASH
<b>p3.4.l</b>	33	4	17.5	310	4	<b>380</b>	119.408	<b>380</b>
<b>p3.4.n</b>	33	4	20	310	5	<b>440</b>	725.713	<b>440</b>
<b>p6.3.i</b>	64	3	18.3	420	15	<b>642</b>	484.208	<b>642</b>
<b>p5.3.k</b>	66	3	18.3	325	12	<b>495</b>	488.751	<b>495</b>
<b>p5.2.l</b>	66	2	30	395	32	<b>800</b>	7141.79	<b>800</b>
<b>p4.4.f</b>	100	4	25	213	13	<b>324</b>	47.5021	<b>324</b>
<b>p4.3.e</b>	100	3	30	235	17	<b>468</b>	3240.48	<b>468</b>
<b>p7.2.f</b>	102	2	60	324	13	362	9891.643	<b>387</b>

Como se pode observar, o Algoritmo Evolutivo, na maioria dos casos, converge para a melhor solução conhecida na literatura. É importante destacar que, mesmo partindo de soluções iniciais com GAP elevado em relação à melhor

solução encontrada na literatura, essa abordagem se mostrou capaz de, em poucas iterações, convergir para soluções ótimas ou próximas da ótima. Como exemplo, pode-se observar na instância *p5.2.l* que o GAP da solução inicial (325) para o valor da melhor solução conhecida era de 59.37%. Apesar de partir de uma solução muito distante da melhor conhecida, em cerca de 2h o algoritmo consegue convergir para a melhor solução com valor de 800.

A Tabela 5.2 retrata, de forma mais detalhada, todas as iterações do Algoritmo Evolutivo executado para a instância *p6.3.i*. As primeiras duas linhas da tabela consistem em um cabeçalho que traz o nome da instância, o número de clientes  $n$ , o número de veículos  $m$  e o valor da melhor solução conhecida *BestSol*. Essa tabela é composta pelos seguintes campos:

- Iter: número da iteração do Algoritmo Evolutivo. No caso da primeira linha, trata-se dos dados da execução do nó raiz apenas;
- $S_1$  : valor da solução base  $s_1$ ;
- $S_2$  : valor da solução base  $s_2$ ;
- $|V^1|$ : número de pontos visitados na solução  $s_1$ ;
- $|V^2|$ : número de pontos visitados na solução  $s_2$ ;
- $|V^1 \cap V^2|$ : cardinalidade da intersecção dos conjuntos de pontos visitados por  $s_1$  e  $s_2$ ;
- $|A^1|$ : número de arcos da solução  $s_1$ ;
- $|A^2|$ : número de arcos da solução  $s_2$ ;
- $|A^1 \cup A^2|$ : cardinalidade da intersecção dos conjuntos de arcos de  $s_1$  e de  $s_2$ ;
- $|V^1 \cup V^2| + |A^1 \cup A^2|$ : número total de variáveis de visitação de vértices e de utilização de arcos;
- $K$ : parâmetro da busca elipsoidal;
- CG UB: limite superior da iteração obtido pela relaxação linear resolvida por geração de colunas.;
- Min Cut: limite superior obtido com os cortes *min cut*;
- BCP : novo limite superior obtido após execução do BCP;
- $S_3$ : nova solução inteira  $s_3$  encontrada pela busca elipsoidal;
- Method: indica se a solução  $s_3$  foi encontrada por heurística que usa informações da solução fracionária ou se  $s_3$  foi encontrada pelo próprio algoritmo BCP;
- $|A^3 \cap A^1|$ : número de arcos herdados por  $s_3$  somente de  $s_1$ ;

- $|A^3 \cap A^2|$ : número de arcos herdados por  $s_3$  somente de  $s_2$ ;
- $|A^1 \cap A^2 \cap A^3|$ : número de arcos herdados por  $s_3$  que pertencem às 2 soluções  $s_1$  e  $s_2$ ;
- $|V^3 - (V^1 \cup V^2)|$ : número de vértices visitados por  $s_3$  que não pertencem nem a  $s_1$  nem a  $s_2$ ;
- Time: tempo computacional da iteração em segundos;
- *GapIter*: representa o GAP entre o limite superior obtido na iteração e a solução inteira encontrada, ou seja, se esse gap for zero significa que foi encontrada a solução ótima da busca local referente a essa iteração ;
- *GapUB*: GAP da solução inteira encontrada na iteração em relação ao limite superior da instância, ou seja, se esse gap for zero, a solução inteira é a solução ótima da instância;

A Tabela 5.2 mostra que mesmo sem realizar a busca elipsoidal na otimalidade, o algoritmo consegue convergir para a solução ótima em 47 iterações. O parâmetro da busca,  $k$ , variou de 14 a 20. O valor inicial do parâmetro é 14 e sempre que não é encontrada nenhuma solução melhor do que as soluções base, aumenta-se a vizinhança. Na última iteração a solução ótima é encontrada e herda muitas características das soluções base. A otimalidade, nesse caso, como também em outros casos que serão apresentados no capítulo 6, é provada pois o limite superior obtido no nó raiz (ver primeira linha da tabela) é igual ao valor da solução encontrada na última iteração (642). O algoritmo pára quando essa condição é alcançada ou quando expira o tempo máximo pré-estabelecido para a execução da instância.

### 5.6.1

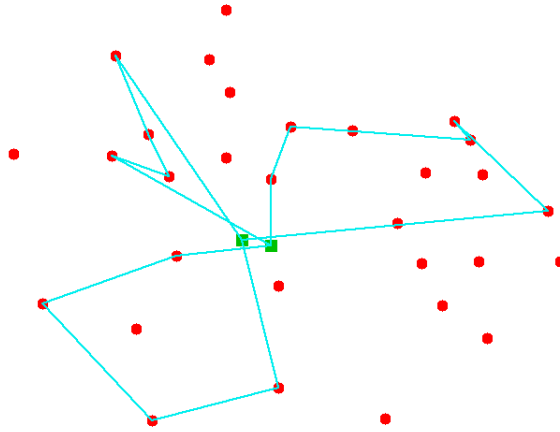
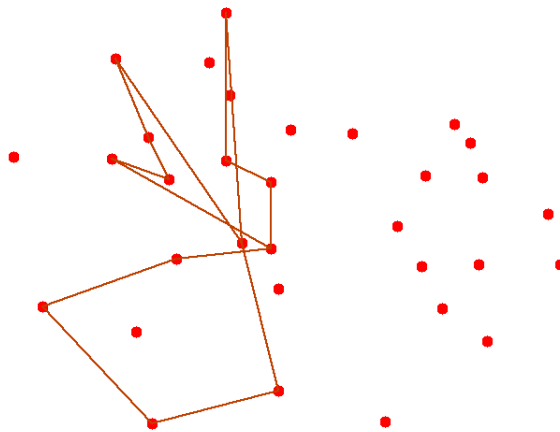
#### Exemplo ilustrado

Essa subseção descreve, de maneira ilustrada, como uma solução  $s_3$  é gerada a partir de duas soluções iniciais  $s_1$  (Figura 5.5) e  $s_2$  (5.6) de uma busca elipsoidal. Neste exemplo, foi utilizada a instância *p1.3.p* que contém 32 pontos e uma frota de 3 veículos.

Seja  $Z_1$  o prêmio total coletado pela solução  $s_1$  e  $Z_2$  o prêmio total coletado pela solução  $s_2$ , onde  $Z_1 = 145$  e  $Z_2 = 125$ .

A Figura 5.7 retrata as semelhanças bem como as diferenças entre as duas soluções base  $s_1$  e  $s_2$ . Os arcos na cor azul escuro pertencem a ambas as soluções base ( $s_1 \cap s_2$ ). Os arcos na cor vermelha pertencem exclusivamente à solução  $s_1$  ( $s_1 - s_2$ ) e os arcos na cor azul claro pertencem somente à solução  $s_2$  ( $(s_2 - s_1)$ ).



Figura 5.5: Solução  $s_1$ Figura 5.6: Solução  $s_2$ 

A partir dessas duas soluções base  $s_1$  e  $s_2$ , foi executada uma busca elipsoidal cujo valor do parâmetro  $k$  que indica o tamanho da vizinhança a ser explorado foi 20.

A Figura 5.8 destaca o conjunto de arcos ( $A_1$ ) provenientes de  $s_1$  que pertencem também à solução resultante da busca elipsoidal  $s_3$ . Os arcos de  $s_3$  que faziam parte da solução  $s_2$ , formam, por sua vez, o conjunto  $A_2$ . Os arcos do conjunto  $A_1$  estão na cor azul escuro ao passo que os arcos do conjunto  $A_2$  estão na cor azul claro.

A Figura 5.9 mostra a solução  $s_3$  completa, com todos os seus arcos, sendo os arcos na cor verde arcos que não pertenciam nem à solução  $s_1$  nem à solução  $s_2$ . Cerca de 25% dos arcos da solução resultante  $s_3$  são provenientes de uma das soluções base ( $s_1$  e  $s_2$ ). O prêmio total coletado pela solução  $s_3$  ( $Z_3$ ) foi igual a 220. Portanto, a solução resultante da busca elipsoidal foi superior às soluções base.

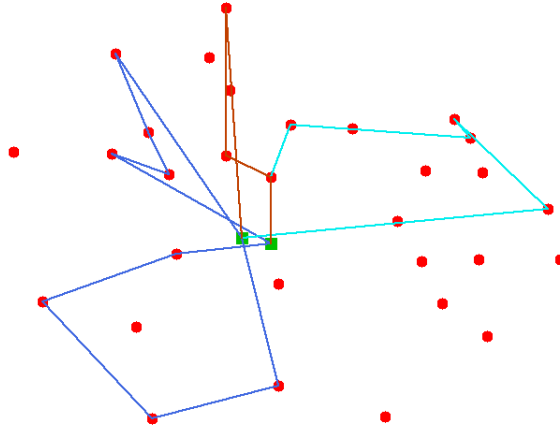


Figura 5.7: Arcos das soluções  $s_1$  e  $s_2$

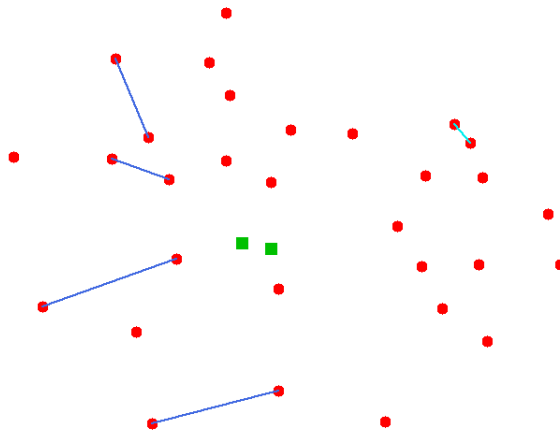
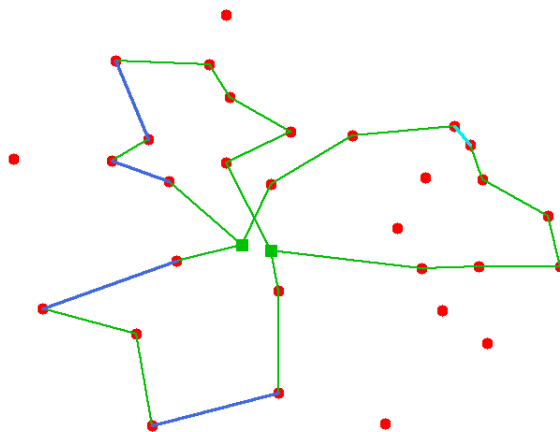
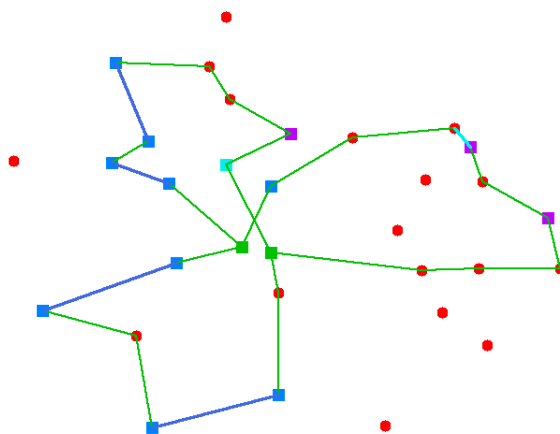


Figura 5.8: Arcos das soluções originais que permanecem na solução  $s_3$

A Figura 5.10 destaca a origem dos pontos da solução  $s_3$ . Os pontos na cor azul escuro pertencem tanto à solução  $s_1$  como à solução  $s_2$  ( $s_1 \cap s_2$ ). Os pontos que pertenciam somente a uma das duas soluções base ( $s_1$  ou  $s_2$ ) foram marcados com a cor roxa. Finalmente, os pontos em vermelho que são visitados por  $s_3$  não pertenciam a nenhuma das soluções base. Vale destacar que cerca de 66% dos pontos da solução resultante da busca elipsoidal foram herdados de pelo menos um das soluções base.

## 5.7 Conclusão

Neste capítulo foi apresentada, resumidamente, a taxonomia proposta por Puchinger e Raidl (PR05) para classificar os algoritmos que combinam técnicas exatas e heurísticas na resolução de problemas de Otimização Combinatória. As vizinhanças circulares e elipsoidais não haviam sido testadas na

Figura 5.9: Solução  $s_3$ Figura 5.10: Origem dos pontos da solução  $s_3$ 

literatura, dentro do esquema de um algoritmo *Branch-Cut-and-Price*. Os resultados mostraram que essas buscas incorporadas a um Algoritmo Evolutivo foram capazes de convergir para soluções ótimas ou próximas da ótima para o TOP. O capítulo 6 irá trazer mais resultados para essa abordagem, a qual será comparada com as melhores metaheurísticas da literatura.

