4
Um algoritmo genético com reconexão por caminhos

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 [68], 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 [44] 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 [14, 54, 55, 73], apesar de não ter sido este o propósito original que levou ao seu desenvolvimento.

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 [30, 31, 32, 33, 59, 66].

Os algoritmos genéticos e a técnica de reconexão por caminhos serão discutidos com maiores detalhes nas Seções 4.1 e 4.2, respectivamente. Nas seções seguintes serão apresentadas as etapas que compõem o algoritmo genético híbrido proposto para o problema da filogenia. Por fim, serão mostrados os resultados computacionais.

4.1 Algoritmos genéticos

Um algoritmo genético é um algoritmo probabilístico que mantém uma população de indivíduos $Pop(t) = \{x_1^t, ..., x_{tamPop}^t\}$ 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 [50], esta busca requer um equilíbrio entre dois objetivos aparentemente conflitantes: o aproveitamento das melhores soluções e a exploração do espaço de busca (explotação × 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 multidirecional, 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.

A Figura 4.1 apresenta a estrutura de um algoritmo genético padrão. Na linha 1, o custo da melhor solução encontrada até o momento é definido como infinito. No laço nas linhas 2-3, a população inicial é construída. No laço nas linhas 4-13, a cada geração a população é submetida à ação de operadores genéticos que modificam a população. Esta nova população é avaliada e, se existir alguma solução melhor do que a melhor obtida até o momento, solução x^*, então esta última é atualizada. Na linha 14 é retornada a melhor solução obtida pelo algoritmo genético.

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

```
Procedimento AG_Padrão (tamPop, maxIterações)
Entrada
    tamPop - tamanho da população;
    maxIterações - número total de iterações (gerações);
Saída
    x* - a melhor solução obtida;
Início
01.   f* ← ∞;
02.   Para i=1, ..., tamPop faça
03.       Construa cada indivíduo i da população;
04.   Para j=1, ..., maxIterações faça
05.   Início
06.       Aplique os operadores genéticos sobre a população de indivíduos;
07.       Avalie a nova população;
08.       Se existir na nova população uma solução x com custo inferior a f* então
09.       Início
10.           x* ← x;
11.           f* ← o custo da solução x;
12.       Fim-se
13.   Fim-para
14.   Retorne x*;
Fim-AG_Padrão
```

Figura 4.1: Algoritmo genético padrão.

(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.).

Nas próximas subseções será explicado detalhadamente cada componente citado acima.

4.1.1 Codificação de indivíduos

Cada indivíduo de uma população representa um candidato em potencial à solução do problema em questão. No algoritmo genético clássico, proposto por Holland [44], as soluções candidatas são codificadas em arranjos binários de tamanho fixo.

Entretanto, em diversas aplicações práticas a utilização de codificação binária leva a um desempenho insatisfatório. Em [50] é argumentado que a representação binária apresenta desempenho pobre quando aplicada a problemas numéricos com alta dimensionalidade.

Outra representação bastante utilizada é a representação por inteiros [14, 50, 54, 55, 73], onde cada gene de um cromossomo pode assumir um valor inteiro qualquer e não somente 0 e 1 como na representação binária. Isso torna esta representação frequentemente mais eficiente do que a representação binária para problemas onde as duas representações são adequadas.

A argumentação feita em [50], de que o desempenho de um algoritmo genético com codificação binária é pobre quando o espaço de busca é de dimensão elevada, não é universalmente aceita na literatura referente a algoritmos genéticos. Em [26] é argumentado que o espaço de busca por si só (sem levar em conta a escolha da representação) não determina a eficiência do algoritmo genético. Espaços de busca de dimensão elevada podem às vezes ser explorados eficientemente, enquanto que espaços de busca de dimensão reduzida podem apresentar dificuldades significativas.

Fica claro, portanto, que a codificação é uma das etapas mais críticas na definição de um algoritmo genético. A definição inadequada da codificação pode levar a problemas de convergência prematura do algoritmo genético. A estrutura de um cromossomo deve representar uma solução como um todo e deve ser a mais simples possível. Em problemas de otimização restrita, a codificação adotada pode fazer com que indivíduos modificados por cruzamento/mutação sejam inválidos. Nestes casos, cuidados especiais devem ser tomados na definição da codificação e/ou dos operadores.

4.1.2 Definição da população inicial

O método mais comumente utilizado na criação da população é a inicialização aleatória dos indivíduos. Se algum conhecimento inicial a respeito do problema estiver disponível, pode ser utilizado na inicialização da população. Por exemplo, se é sabido que a solução final (assumindo codificação binária) vai apresentar mais 0's do que 1's, então esta informação pode ser utilizada, mesmo que não se saiba exatamente a proporção. Já em problemas com restrição, deve-se tomar cuidado para não gerar indivíduos inválidos na etapa de inicialização.

Entretanto, em muitas aplicações a utilização do método de criação aleatória gera uma população com indivíduos de pouca qualidade, o que exige dos operadores genéticos muito esforço (muitas gerações) para atingir uma boa solução. É mais eficiente usar heurísticas construtivas aleatorizadas para a criação da população inicial.

4.1.3
Função de avaliação de cromossomos

A função de avaliação tem o papel de simular a pressão exercida pelo ambiente sobre o indivíduo. Esta função deve indicar a qualidade de cada cromossomo na população. Em otimização, ela está intimamente ligada à função objetivo que se deseja minimizar ou maximizar.

4.1.4
Operadores genéticos

Os operadores genéticos mais freqüentemente utilizados em algoritmos genéticos são o cruzamento e a mutação. Nesta subseção serão apresentados os principais aspectos relacionados a estes operadores.

Operador de Cruzamento

O operador de cruzamento ou recombinação cria novos indivíduos através da combinação de dois ou mais indivíduos. A idéia intuitiva por trás do operador de cruzamento é a troca de informação entre diferentes soluções candidatas. No algoritmo genético clássico é atribuída uma probabilidade de cruzamento fixa aos indivíduos da população.

O operador de cruzamento mais comumente empregado é o cruzamento de um ponto. Para a aplicação deste operador, são selecionados dois indivíduos (pais) e a partir de seus cromossomos são gerados dois novos indivíduos (filhos). Para gerar os filhos, seleciona-se um ponto de corte aleatoriamente nos cromossomos dos pais e os segmentos de cromossomo criados a partir do ponto de corte são trocados. Para um melhor entendimento, é apresentado a seguir um exemplo deste operador, onde p_1 e p_2 são os cromossomos pais e os cortes são representados pelo caracter "|". Neste exemplo, os cortes foram feitos no meio do cromossomo, o que não precisa ocorrer sempre.

$$p_1 = (1\ 0\ 1 \mid 0\ 1\ 0) = (p_{11} \mid p_{21})$$
$$p_2 = (1\ 1\ 1 \mid 0\ 0\ 0) = (p_{12} \mid p_{22})$$

O cromossomo p_1 é composto por duas partes, p_{11} e p_{21}, onde p_{11} é a parte do cromossomo p_1 antes do corte e p_{21} a parte depois do corte. Do mesmo modo, o cromossomo p_2 é composto por p_{12} e p_{22}. Trocando-se, entre estes cromossomos, as partes p_{21} e p_{22}, obtém-se os cromossomos filhos O_1 e O_2 como descrito a seguir.

$$O_1 = (p_{11} \mid p_{22}) = (1\ 0\ 1 \mid 0\ 0\ 0)$$
$$O_2 = (p_{12} \mid p_{21}) = (1\ 1\ 1 \mid 0\ 1\ 0)$$

Muitos outros tipos de cruzamento têm sido propostos na literatura [16, 50, 72, 76]. Uma extensão simples do cruzamento de um ponto é o cruzamento de dois pontos, onde dois pontos de corte são escolhidos e material genético são trocados entre eles. Outro tipo de cruzamento muito comum é o cruzamento uniforme [72]: para cada gene no primeiro filho é decidido (com alguma probabilidade fixa p) qual pai vai contribuir com seu valor para aquela posição. Como o cruzamento uniforme troca genes ao invés de segmentos de genes, ele pode combinar características independentemente da sua posição relativa no cromossomo.

Em [16] são relatados diversos experimentos com vários operadores de cruzamento. Os resultados indicam que o operador com pior desempenho é o cruzamento de um ponto; entretanto, não há nenhum operador de cruzamento que claramente apresente um desempenho superior aos demais. Uma conclusão a que se pode chegar a partir destes resultados é que cada operador de cruzamento é particularmente eficiente para uma determinada classe de problemas e extremamente ineficiente para outras.

Operador de mutação

O operador de mutação modifica aleatoriamente um ou mais genes de um cromossomo. A probabilidade de ocorrência de mutação em um gene é denominada *taxa de mutação*. Usualmente, são atribuídos valores pequenos para a taxa de mutação. A idéia intuitiva por trás do operador de mutação é criar uma variabilidade extra na população, mas sem destruir o progresso já obtido com a busca.

Considerando codificação binária, o operador de mutação padrão simplesmente troca o valor de um gene em um cromossomo [44]. Assim, se um gene selecionado para mutação tem valor 1, o seu valor passará a ser 0 após a aplicação da mutação, e vice-versa. Um exemplo deste tipo de mutação é apresentado na Figura 4.2. Existem alguns outros tipos de operadores de mutação na literatura científica [50, 51].

Antes:	1	1	1	0	0
Depois:	1	1	0	0	0

Figura 4.2: Exemplo de mutação.

4.1.5
Seleção de indivíduos

O algoritmo genético clássico seleciona os indivíduos aleatoriamente [44]. Neste método todos os indivíduos possuem a mesma probabilidade de serem selecionados. Uma outra estratégia de seleção de indivíduos é denominada *roleta russa* [50]. A roleta russa atribui a cada indivíduo de uma população uma probabilidade de ser selecionado proporcional à sua avaliação, em relação à soma da avaliação de todos os indivíduos da população. Assim, quanto melhor a avaliação de um indivíduo, maior a probabilidade dele ser selecionado.

Outro exemplo de mecanismo de seleção é a seleção baseada em classificação [7]. Esta estratégia utiliza as posições dos indivíduos quando ordenados de acordo com a avaliação para determinar a probabilidade de seleção.

4.1.6
Definição dos parâmetros nos algoritmos genéticos

O principal parâmetro a ser definido é o tamanho da população que deve ser utilizada no AG. Testes experimentais mostram que quanto maior for a população, melhores serão os resultados alcançados. Mas, por outro lado, quanto maior a população, maior será também o tempo computacional exigido por geração.

O que é necessário para uma boa definição dos parâmetros é realizar um estudo com experimentos variando o valor de cada parâmetro. Tenta-se com isso definir a combinação de parâmetros mais apropriada para o problema em questão.

4.2
Reconexão por caminhos

A reconexão por caminhos incorporada a um algoritmo GRASP tem como objetivo intensificar a busca em regiões onde soluções de qualidade tenham sido encontradas. A reconexão por caminhos foi inicialmente introduzida no contexto da metaheurística busca tabu [34, 61], como uma abordagem para integrar estratégias de intensificação e diversificação ao processo de busca. Em [35], é apresentado um *survey* sobre reconexão por caminhos. A técnica consiste em explorar trajetórias que conectam soluções de alta qualidade, começando de uma *solução inicial* e gerando um caminho na

vizinhança dessa solução na direção de outra solução, chamanda de *solução guia.* Esse caminho é gerado selecionando-se movimentos que introduzam atributos da solução guia na solução inicial. A cada passo, todos os movimentos que incorporam atributos da solução guia são analisados e o melhor movimento é escolhido.

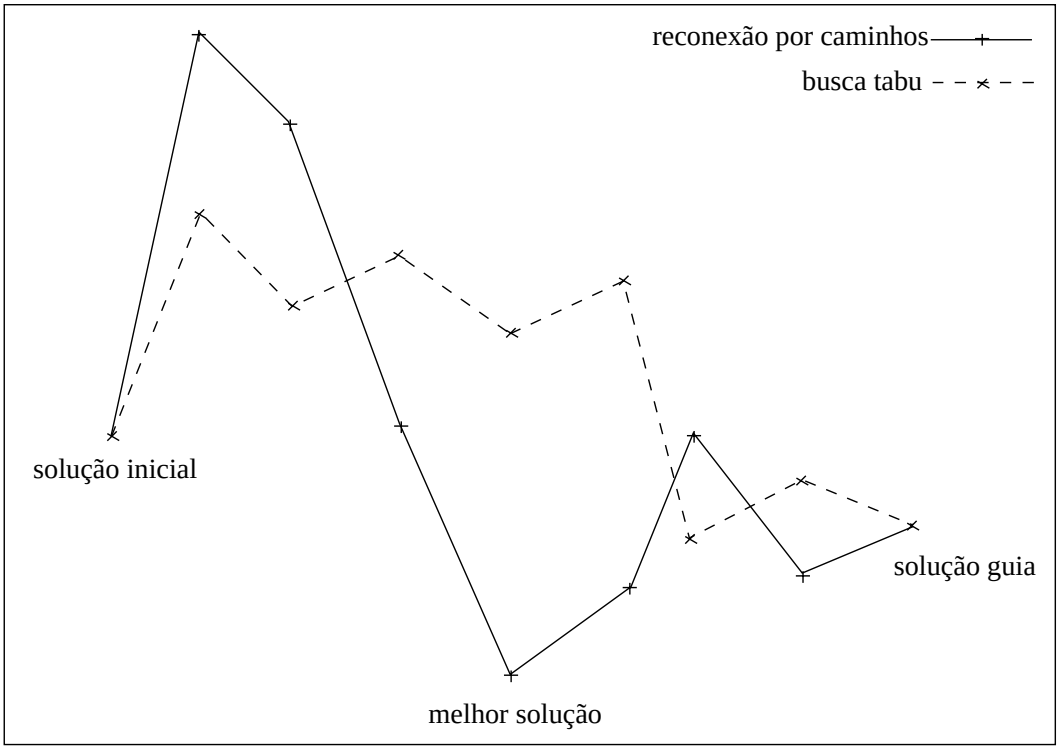

Figura 4.3: Reconexão por caminhos no contexto da busca tabu.

Essa abordagem é denominada de reconexão por caminhos porque quaisquer duas soluções visitadas por um algoritmo de busca tabu estão ligadas por uma seqüência de movimentos. Na Figura 4.3 são mostrados dois caminhos hipotéticos, que ligam uma solução inicial a uma solução guia através de uma seqüência de movimentos. A linha pontilhada mostra soluções visitadas por um algoritmo de busca tabu e a linha contínua mostra as soluções visitadas pela reconexão por caminhos. Observa-se que as trajetórias seguidas pelas duas estratégias são diferentes. Isso ocorre principalmente porque, em cada iteração do algoritmo de busca tabu, os movimentos são escolhidos de forma "gulosa", isto é, escolhe-se o movimento não proibido que minimize localmente o valor da função objetivo. Durante a reconexão por caminhos, o objetivo principal é incorporar atributos da solução guia à solução inicial. Como apresentado na Figura 4.3, o objetivo de realizar a reconexão por caminhos no contexto da metaheurística busca

```
Procedimento ReconexãoPadrão (x_inicial, x_guia)
Entrada
    x_inicial - a solução de onde começa o caminho;
    x_guia - a solução onde termina o caminho;
Saída
    x* - a melhor solução obtida;
Início
01.   f* ← ∞;
02.   x ← x_inicial;
03.   Enquanto x ≠ x_guia faça
04.   Início
05.       Realize um movimento que torne a solução x mais próxima de x_guia;
06.       Avalie a nova solução x;
07.       Se f(x) < f* então
08.       Início
09.           x* ← x;
10.           f* ← f(x);
11.       Fim-se
12.   Fim-enquanto
13.   Retorne x*;
Fim-ReconexãoPadrão
```

Figura 4.4: Algoritmo de reconexão por caminhos padrão.

tabu é obter soluções não visitadas pela trajetória original, que melhorem a função objetivo.

A Figura 4.4 apresenta o algoritmo de reconexão por caminhos padrão. Este recebe como parâmetros de entrada a solução inicial, $x_{inicial}$, e a solução guia, x_{guia}. A solução x, que guarda a cada iteração a solução corrente no caminho entre $x_{inicial}$ e x_{guia}, é inicializada com $x_{inicial}$ na linha 2. O laço nas linhas 3-12 realiza os movimentos que transformarão a solução x em x_{guia}. Nas linhas 7-11 é verificado se a melhor solução no caminho, x^*, deve ser atualizada. Finalmente, na linha 13 a melhor solução é retornada.

4.3 Criação da população inicial

Na fase inicial do algoritmo genético, uma população de $tamPop = 100$ indivíduos é gerada através de um algoritmo de construção randomizado. Usou-se nesta etapa o algoritmo `GStep_wR` [3, 4] descrito na Seção 3.3 para a construção de cada indivíduo da população, pois os estudos realizados em [3, 4] mostraram que este encontra, geralmente, as melhores soluções (embora o esforço computacional exigido seja mais elevado quando comparado com outros algoritmos).

O algoritmo de construção da população inicial é apresentado na Figura 4.5. O laço nas linha 1-2 faz com que cada indivíduo da população

Procedimento `ConstruirPopulação` (*tamPop*)
Entrada
tamPop - tamanho da população;
Saída
Pop - população construída;
Início
01. **Para** *i*=1, ..., *tamPop* **faça**
02. Aplique o algoritmo `GStep_wR` para construir o *i*-ésimo indivíduo de *Pop*;
03. **Retorne** *Pop*;
Fim-`ConstruirPopulação`

Figura 4.5: Algoritmo de construção da população inicial.

seja construído utilizando o algoritmo de construção aleatorizado `GStep_wR`. Na linha 3 é retornada a população criada.

4.3.1 Evolução da população

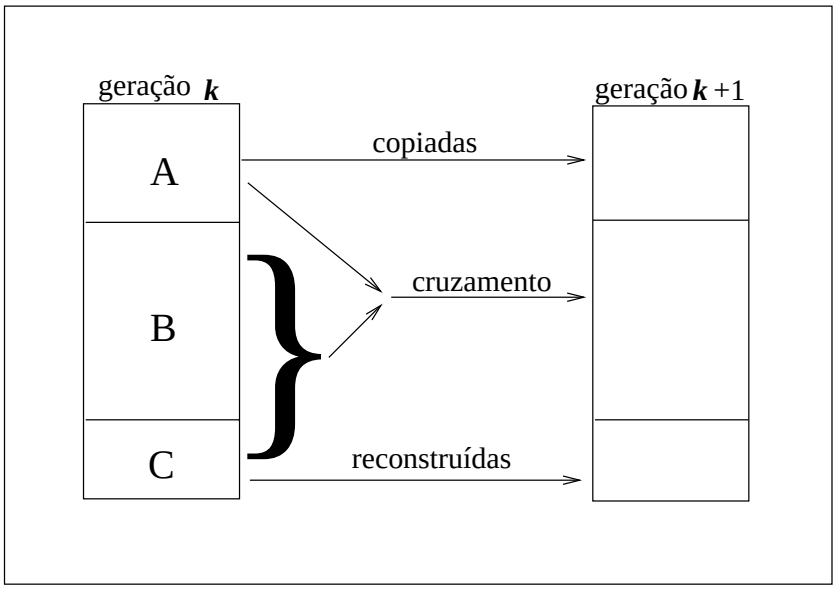

Figura 4.6: Evolução da população.

Para a evolução da população seguiu-se a estratégia proposta em [8] e já usada com sucesso em [10]. Nela, a evolução da população ao longo das gerações é realizada como mostrado na Figura 4.6. A cada geração k a população é ordenada e dividida em classes. A classe A é composta pelos melhores indivíduos, que representam 30% da população. A classe C pelos piores indivíduos, que representam 20% da população. A classe B é formada pelos indivíduos restantes. Para gerar a população da geração k+1, a partir da geração k, os seguintes passos são executados.

1. Todos os indivíduos da classe A são copiados para a próxima geração.

2. Todos os elementos da classe C são reconstruídos, ou seja, 20% dos indivíduos de cada geração são construídos utilizando o algoritmo de construção aleatorizado `GStep_wR` [3, 4] descrito na Seção 3.3. É o mesmo procedimento utilizado na criação da população inicial.

3. Os indivíduos restantes são provenientes de um cruzamento entre um representante escolhido aleatóriamente na classe A com um representante também escolhido aleatoriamente nas classes B ou C. A cada *freqBuscaLocal* gerações uma busca local é aplicada aos indivíduos provenientes deste cruzamento para refinar a qualidade da população. A busca local implementada utiliza a vizinhança SPR (*Subtree Pruning and Regrafting*) descrita na seção 3.4, cuja a implementação foi otimizada em [65]. 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 a 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 freqüência desta busca local é definida de acordo com o método utilizado no cruzamento de indivíduos. Os métodos de cruzamento serão discutidos na Seção 4.5.

A estrutura do algotitmo genético implementado neste trabalho é apresentada na Figura 4.7. Na linha 4, a população inicial, *Pop*, é construída. No laço nas linhas 5-30, esta população é evoluída até atingir um número máximo de renovações, *maxRenovações* [14, 54, 73]. Uma população é submetida a uma renovação quando a qualidade da classe elite, classe A, não melhora entre duas gerações consecutivas. Isso é verificado no algoritmo nas linhas 7-17. Como medida de qualidade, usou-se o somatório dos valores de parcimônia de cada indivíduo pertencente à classe A. A estratégia de renovação da população (algoritmo `RenovarPopulação`) será discutida com maiores detalhes na Seção 4.4. Na linha 18, a classe elite A da geração k é copiada para a geração k+1. Na linha 19, a classe C é completamente reconstruída para a geração k+1. O restante da população da geração k+1 é definida nas linhas 20-28 utilizando cruzamentos genéticos entre representantes da classe A e representantes das classes B ou C. Este cruzamento será feito via reconexão por caminhos, o qual será discutido com maiores

Procedimento `EvoluirPopulação` ($tamPop$, $freqBuscaLocal$, $maxRenovações$)
Entrada
$tamPop$ - tamanho da população;
$freqBuscaLocal$ - freqüência da busca local;
$maxRenovações$ - número de renovações a serem realizadas;
Saída
s^* - melhor solução obtida;
Início
01. $numRenovações \leftarrow 0$;
02. $k \leftarrow 1$;
03. $qualidade \leftarrow \infty$;
04. $Pop \leftarrow$ `ConstruirPopulação` ($tamPop$);
05. **Enquanto** $numRenovações < maxRenovações$ **faça**
06. **Início**
07. Ordene a população Pop crescentemente pelo valor de parcimônia dos indivíduos;
08. $novaQualidade \leftarrow \sum_l f(I_l)$, onde I_l é o l-ésimo indivíduo de Pop, $f(I_l)$ é o valor de parcimônia do indivíduo I_l e $1 \leq l \leq 0.3 \times tamPop$;
09. **Se** $qualidade = novaQualidade$ **então**
10. **Início**
11. $Pop \leftarrow$ `RenovarPopulação`(Pop);
12. $numRenovações \leftarrow numRenovações + 1$;
13. $qualidade \leftarrow \infty$;
14. Ordene a população Pop crescentemente;
15. **Senão**
16. $qualidade \leftarrow novaQualidade$;
17. **Fim-se**
18. Copie os 30% primeiros indivíduos da população para geração k+1;
19. Construa, com o algoritmo `GSTep_wR`, 20% da população da geração k+1;
20. **Para** cada indivíduo i da geração k+1 ainda não definido **faça**
21. **Início**
22. Escolha aleatoriamente um representante da classe A, $pai1$;
23. Escolha aleatoriamente um representante da classe B ou C, $pai2$;
24. $filho \leftarrow$ `RealizarCruzamento`$(pai1, pai2)$;
25. **Se** k é múltiplo de $freqBuscaLocal$ **então**
26. $filho \leftarrow \mathtt{BL}^1(filho)$;
27. Adicione $filho$ como o i-ésimo indivíduo da geração $k+1$;
28. **Fim-para**
29. $k \leftarrow k + 1$;
30. **Fim-enquanto**
31. Ordene a população Pop crescentemente pelo valor de parcimônia dos indivíduos;
32. $s^* \leftarrow$ o primeiro indivíduo da população pop;
33. **Retorne** s^*;
Fim-`EvoluirPopulação`

Figura 4.7: Estrutura do algoritmo genético implementado.

Procedimento `RenovarPopulação`(Pop)
Entrada
Pop - a população a ser renovada;
Saída
Pop' - a população renovada;
Início
01. Copie os 3% primeiros indivíduos de Pop para Pop';
02. Construa, com o algoritmo `GSTep_wR`, o restante da população Pop';
03. **Retorne** Pop';
Fim-`RenovarPopulação`

Figura 4.8: Algoritmo de renovação da população.

detalhes na Seção 4.5, juntamente com o algoritmo `RealizarCruzamento`. Na linha 26 é aplicada uma busca local utilizando a vizinhança SPR a cada *freqBuscaLocal* gerações. Assim, é dada uma contribuição para que a população evolua mais depressa. Por fim, na linha 33 é retornada a melhor solução, s^*, encontrada pelo algoritmo.

4.4 Renovação da população

A renovação de uma população é realizada quando algum critério utilizado 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, por algum critério, são consideradas importantes.

A estratégia usada neste trabalho foi a de manter uma parte da classe elite e reconstruir os indivíduos restantes. Assim, obtem-se uma diversidade na população e, mantendo os melhores indivíduos, também mantem-se um material genético importante para as próximas gerações. A Figura 4.8 apresenta o algoritmo usado para a renovação da população. Os 3% melhores indivíduos de Pop são copiados para Pop' na linha 1. Na linha 2, o restante dos indivíduos de Pop' são construídos usando o algoritmo `GStep_wR` [3, 4] descrito na seção 3.3. Finalmente na linha 3 a nova população é retornada.

Os resultados que serão apresentados na Seção 4.6 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.

4.5 Cruzamento de indivíduos via reconexão por caminhos

A técnica de reconexão por caminhos é uma estratégia de intensificação originalmente proposta por Glover [33] para explorar trajetórias entre soluções elites obtidas pela busca tabu ou pela heurística busca espalhada. Usando uma ou mais soluções elites, caminhos no espaço de soluções guiados por outras soluções elites são explorados na busca por soluções melhores. Para gerar caminhos, os movimentos são selecionados com a finalidade de introduzir atributos na solução atual que aparecem na solução elite guia.

Extensões, melhoramentos, e aplicações eficientes de reconexão por caminhos no contexto de implementações GRASP são relatadas na literatura [60, 61, 62]. Neste trabalho é realizado uma operação de cruzamento usando uma estratégia de reconexão por caminhos. Dadas duas soluções s_1 e s_2, uma reconexão por caminhos bidirecional é realizado entre elas [60] e a melhor solução encontrada é retornada.

Este mecanismo é uma extensão da operação de cruzamento tradicional. Ao invés de produzir apenas um filho, este investiga um grupo de soluções compartilhando as características dos pais. A solução encontrada pela reconexão por caminhos é o melhor filho que pode ser obtido por uma operação de cruzamento convencional.

Sejam s_1 e s_2, respectivamente, a filigenia inicial e a filogenia guia para a reconexão por caminhos. Sejam N_1 um nó de s_1, e N_2 o seu respectivo em s_2. O objetivo da reconexão por caminhos é fazer com que, após um certo número de iterações, o conjunto de taxons operacionais existentes na subárvore filha esquerda de N_1 seja igual ao conjunto de taxons operacionais existentes na subárvore esquerda de N_2. Consequentemente, isso faz com que o conjunto de taxons operacionais na subárvore direita de N_1 seja igual ao conjunto da subárvore direita de N_2. Esse processo é iniciado na raiz de s_1 e propagado até suas folhas.

A Figura 4.9 apresenta o algoritmo de cruzamento de indivíduos utilizando reconexão por caminhos. O laço nas linhas 2-37 garante que o reconexão por caminhos aplicado será bidirecional. O laço nas linhas 11-36 faz com que todos os nós da solução inicial, $s_{inicial}$, sejam submetidos à análise, começando da raiz e indo até as folhas. O trecho nas linhas 20-23 verifica se o lado esquerdo de N_1 se parece mais com o lado esquerdo ou direito de N_2. Se for mais semelhante ao lado direito, é realizada na linha 23 uma troca entre a subárvore filha esquerda de N_2 com a subárvore filha direita de N_2. Isso garante que a subárvore esquerda de N_1 estará sempre

Procedimento `RealizarCruzamento`(s_1, s_2)
Entrada
s_1 e s_2 - os indivíduos pais;
Saída
s^* - a melhor solução no caminho de s_1 a s_2 ou no de s_2 a s_1;
Início
01. $f^* \leftarrow \infty$;
02. **Para** i=1, ..., 2 **faça**
03. **Início**
04. **Se** $i = 1$ **então**
05. $s_{inicial} = s_1$; $s_{guia} = s_2$;
06. **Senão**
07. $s_{inicial} = s_2$; $s_{guia} = s_1$;
08. $s \leftarrow s_{inicial}$;
09. Inicialize a pilha Q_1 com raiz de s;
10. Inicialize a pilha Q_2 com raiz de s_{guia};
11. **Enquanto** $Q_1 \neq \emptyset$ **faça**
12. **Início**
13. Desempilhar N_1 de Q_1 e N_2 de Q_2;
14. Sejam $FEsqN_1$ e $FDirN_1$, os filhos esquerdo e direito de N_1, respectivamente;
15. Sejam $FEsqN_2$ e $FDirN_2$, os filhos esquerdo e direito de N_2, respectivamente;
16. Seja $FolhasFEsqN_1$ o conjunto de taxons operacionais que são folhas na subárvore enraizada pelo nó $FEsqN_1$;
17. Seja $FolhasFDirN_1$ o conjunto de taxons operacionais que são folhas na subárvore enraizada pelo nó $FDirN_1$;
18. Seja $FolhasFEsqN_2$ o conjunto de taxons operacionais que são folhas na subárvore enraizada pelo nó $FEsqN_2$;
19. Seja $FolhasFDirN_2$ o conjunto de taxons operacionais que são folhas na subárvore enraizada pelo nó $FDirN_2$;
20. $Comp1 \leftarrow$ o número de taxons comuns entre $FolhasFEsqN_1$ e $FolhasFEsqN_2$ adicionado do número de taxons comuns entre $FolhasFDirN_1$ e $FolhasFDirN_2$;
21. $Comp2 \leftarrow$ o número de taxons comuns entre $FolhasFEsqN_1$ e $FolhasFDirN_2$ adicionado do número de taxons comuns entre $FolhasFDirN_1$ e $FolhasFEsqN_2$;
22. **Se** $Comp2 > Comp1$ **então**
23. Troque $FEsqN_2$ com $FDirN_2$;
24. Seja W o conjunto dos taxons operacionais mal posicionados em s, ou seja, que deveriam estar na subárvore direita e estão na esquerda, e vice-versa;
25. Escolha $t \in W$ a ser reposicionado.
26. Remova t de s, gerando s';
27. Escolha uma aresta q de s para posicionar t. Se t estava na subárvore esquerda, então escolha q na subárvore direita. Caso contrário, escolha q na subárvore esquerda. Após o reposicionamento a solução s'' é gerada;
28. **Se** $f(s'') < f^*$ **então**
29. $s^* \leftarrow s''$; $f^* \leftarrow f(s'')$;
30. $s \leftarrow s''$;
31. **Se** N_1 não é folha **então**
32. **Início**
33. Empilhar $FEsqN_1$ e $FDirN_1$ em Q_1;
34. Empilhar $FEsqN_2$ e $FDirN_2$ em Q_2;
35. **Fim-se**
36. **Fim-enquanto**
37. **Fim-para**
38. **Retorne** s^*;
Fim-`RealizarCruzamento`

Figura 4.9: Algoritmo de cruzamento de indivíduos utilizando reconexão por caminhos.

sendo comparada com a subárvore esquerda de N_2, e, consequentemente, a subárvore direita de N_1 estará sempre sendo comparada com a subárvore direita de N_2. No trecho nas linhas 24-27, uma folha mal posicionada t de $s_{inicial}$ é desconectada, e em seguida reinserida em uma aresta q. No decorrer desta seção serão apresentadas diferentes estratégias de reconexão por caminhos para o problema da filogenia que se diferem pelo critério de seleção da folha mal-posicionada a ser desconectada e pelo critério usado para reinseri-la. Esse movimento deve tornar a solução $s_{inicial}$ mais próxima de s_{guia}. Nas linhas 30-31, a melhor solução, s^*, é atualizada. Por fim, a solução s^* é retornada na linha 40.

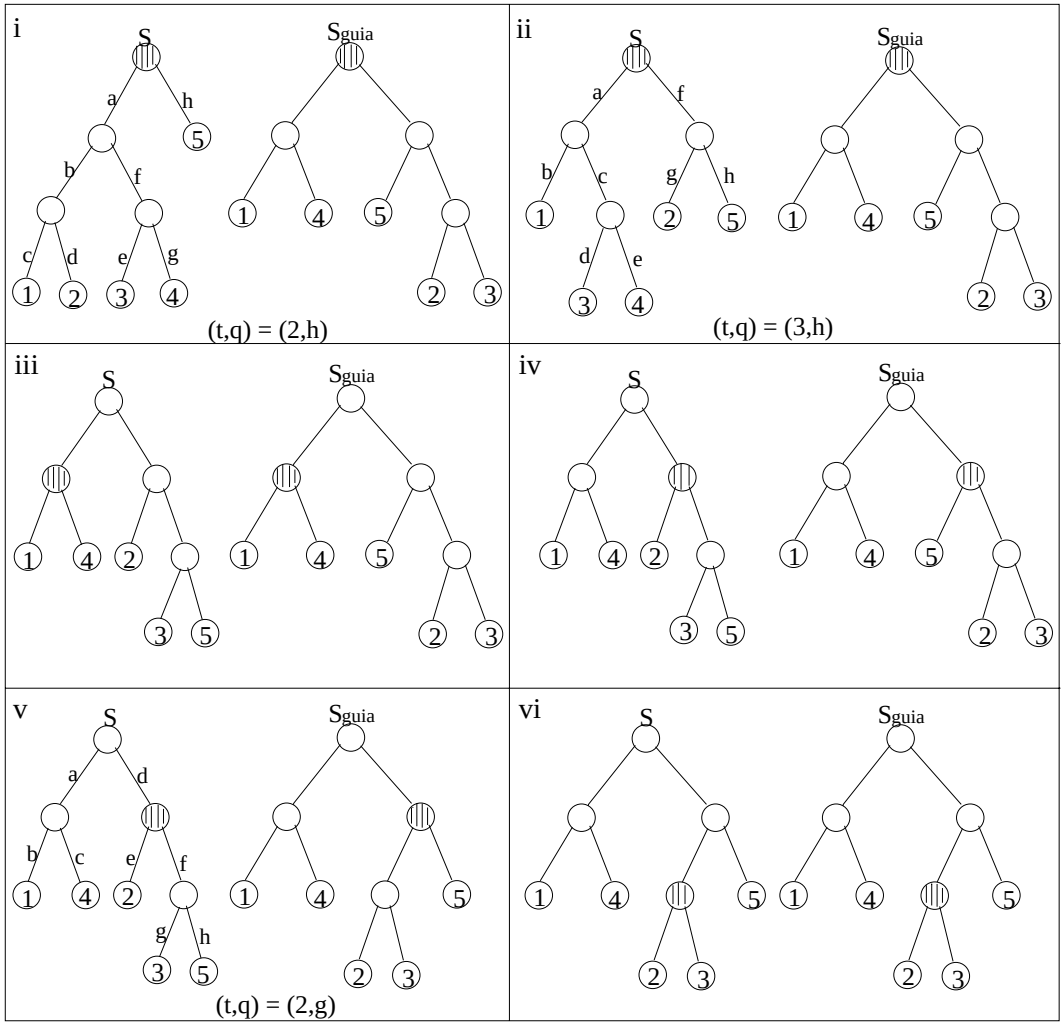

Figura 4.10: Exemplo de reconexão por caminhos em filogenias.

A Figura 4.10 apresenta um exemplo de reconexão por caminhos entre duas filogenias, $s_{inicial}$ e s_{guia}. Na Figura 4.10-i a filogenia s recebe uma cópia de $s_{inicial}$. A filogenia s deve ser modificada até se tornar igual a s_{guia}. O nó hachurado em s recebe a denominação de N_1, enquanto o seu nó equivalente

em s_{guia}, também hachurado, recebe a denominação de N_2. Sejam $FEsqN_1$ e $FDirN_1$ as subárvores esquerda e direita de N_1, respectivamente. Sejam $FEsqN_2$ e $FDirN_2$ as subárvores esquerda e direita de N_2. O primeiro passo a ser realizado, segundo o algoritmo da Figura 4.9, é calcular o valor de $Comp1$ e $Comp2$. $Comp1$ é igual ao número de taxons comuns entre o conjunto das folhas de $FEsqN_1$ e $FEsqN_2$ adicionado ao número de taxons comuns entre o conjunto das folhas de $FDirN_1$ e $FDirN_2$ No exemplo, $Comp1$ é dado por $|\ (\{1,2,3,4\} \cap \{1,4\}) \cup (\{5\} \cap \{2,3,5\})\ |$. Logo, $Comp1 = 3$. $Comp2$ é igual ao número de taxons comuns entre o conjunto das folhas de $FEsqN_1$ e $FDirN_2$ adicionado ao número de taxons comuns entre o conjunto das folhas de $FDirN_1$ e $FEsqN_2$. Logo, $Comp2 = 2$. Sendo $Comp1 \geq Comp2$, então as subárvores $FEsqN_2$ e $FDirN_2$ não precisam ser trocadas. As arestas de s foram identificadas por letras que variam de a a h. A tupla (t, q) (onde t é uma folha mal posicionada que será desconectada e q uma aresta para a reinserção de t) escolhida é (2,h). A nova filogenia s é mostrada na Figura 4.10-ii. Nesta figura, $Comp1$ vale 4 e $Comp2$ vale 1. Novamente, não há necessidade de trocar $FEsqN_2$ com $FDirN_2$. As arestas são renomeadas e a tupla escolhida é $(t, q) =$ (3,h). Na Figura 4.10-iii, o nó N_1 passa a ser o novo nó hachurado em s, pois o processamento já foi finalizado no antigo nó N_1. As variáveis N_2, $FEsqN_1$, $FDirN_1$, $FEsqN_2$ e $FDirN_2$ também são atualizadas de acordo com o novo N_1. Na Figura 4.10-iii há nada a fazer pois não existe folha mal posicionada em relação ao nó N_1. Na Figura 4.10-iv, as variáveis são novamente atualizadas. O valor de $Comp1$ é 1 e o de $Comp2$ é 2. Neste caso, como $Comp1 < Comp2$, as subárvores de N_2, $FEsqN_2$ e $FDirN_2$, são trocadas. A Figura 4.10-v apresenta a filogenia s_{guia} após a modificação. Na Figura 4.10-v, o valor de $Comp1$ é 2 e o de $Comp2$ é 1. As arestas são renomeadas e a tupla escolhida é $(t, q) =$ (2,g). Na Figura 4.10-vi, há mais nada a fazer, pois s é igual a s_{guia}.

Duas estratégias de reconexão por caminhos foram implementadas. O que diferencia as duas são o critério de seleção da folha t mal-posicionada que será desconectada e o critério de seleção da aresta q onde a folha t será reinserida. As duas estratégias são descritas a seguir.

Na primeira, chamada de reconexão por caminhos do tipo 1 (ou simplesmente `PR1`), a folha t mal posicionada a ser desconectada será aquela que apresentar a maior redução de passos evolutivos na desconexão. Como mostrado na Subseção 2.4.2, a redução de passos evolutivos na desconexão de uma subárvore pode ser calculada em tempo $O(m)$ pelo algoritmo `CalcValorReduçãoBin`. Como existem $O(n)$ folhas mal posicionadas a serem testadas, esse passo é realizado em tempo $O(mn)$, onde n é o

número de taxons operacionais e m é o número de características. Uma vez desconectada a folha t, é testada qual a aresta q que produz o menor custo de reinserção e que não deixa t mal posicionada. Como visto na Subseção 2.4.2, cada teste de inserção de uma subárvore pode ser feita em tempo $O(m)$ pelo algoritmo `CalcValorInsercaoBin`. Como existem $O(n)$ arestas possíveis de reinserção, este passo leva tempo $O(mn)$. Logo, o procedimento de encontrar a próxima solução do caminho é realizado em tempo $O(mn)$.

A segunda estratégia, denominada reconexão por caminhos do tipo 2 (ou simplesmente `PR2`), verifica cada par (t, q), onde t é uma folha mal posicionada e q é uma possível aresta para a reinserção de t (que não torna t uma folha mal posicionada), e extrai aquele onde o custo da reinserção menos a redução de passos evolutivos ocasionados pela desconexão produz o menor valor. Como existem $O(n)$ folhas mal posicionadas e $O(n)$ possíveis arestas para reconexão e cada teste de inserção pode ser realizado em $O(m)$ pelo algoritmo `CalcValorInsercaoBin` mostrado na Subseção 2.4.2, o procedimento de encontrar a próxima solução do caminho é realizado em tempo $O(mn^2)$.

4.6 Resultados computacionais

Todos os experimentos computacionais foram realizados em um computador com processador Pentium IV com 2GHz de freqüência e com memória principal de 512MBytes. A heurística `AG+PR1`, que combina o algoritmo `EvoluirPopulação` descrito na Figura 4.7 com a reconexão por caminhos do tipo 1 discutida na Seção 4.5, e a heurística `AG+PR2`, que combina o algoritmo `EvoluirPopulação` descrito na Figura 4.7 com a reconexão por caminhos do tipo 2 discutida na Seção 4.5, foram implementadas em C usando a versão 6.0 do compilador Microsoft Visual C++.

Usou-se uma implementação em C do gerador de números aleatórios descrito em [67].

O algoritmo `AG+PR1` usou uma freqüência de busca local $freqBuscaLocal = 15$ gerações, enquanto o algoritmo `AG+PR2` usou $freqBuscaLocal = 7$ gerações.

O comportamento das novas heurísticas `AG+PR1` e `AG+PR2` foi comparado com o algoritmo `GRASP+VND` [65] apresentado no Capítulo 3 para a instância SCHU (descrita na Tabela 2.1) usando a metodologia proposta por Aiex et al. [2] e recentemente revisado por Resende e Ribeiro [60]. Cem execuções independentes de cada heurística foram feitas. Cada exe-

cução terminava quando uma solução de valor menor ou igual a um certo valor alvo era encontrada. Para o valor alvo foi atribuído o valor da melhor solução previamente conhecida para a instância SCHU, 760. Embora este valor tenha sido escolhido de tal forma que a heurística mais rápida pudesse terminar depois de um tempo computacional considerável, o comportamento relativo das duas heurísticas não é afetado por esta escolha. Distribuições empíricas de probabilidade para a variável tempo-para-valor-alvo são traçadas na Figura 4.11. Para traçar a distribuição empírica para cada algoritmo, seguiu-se o procedimento descrito em [2]. Associou-se com o i-ésimo menor tempo de execução t_i uma probabilidade $p_i = (i - \frac{1}{2})/100$, e marcou-se o ponto $z_i = (t_i, p_i)$, para $i = 1, ..., 100$.

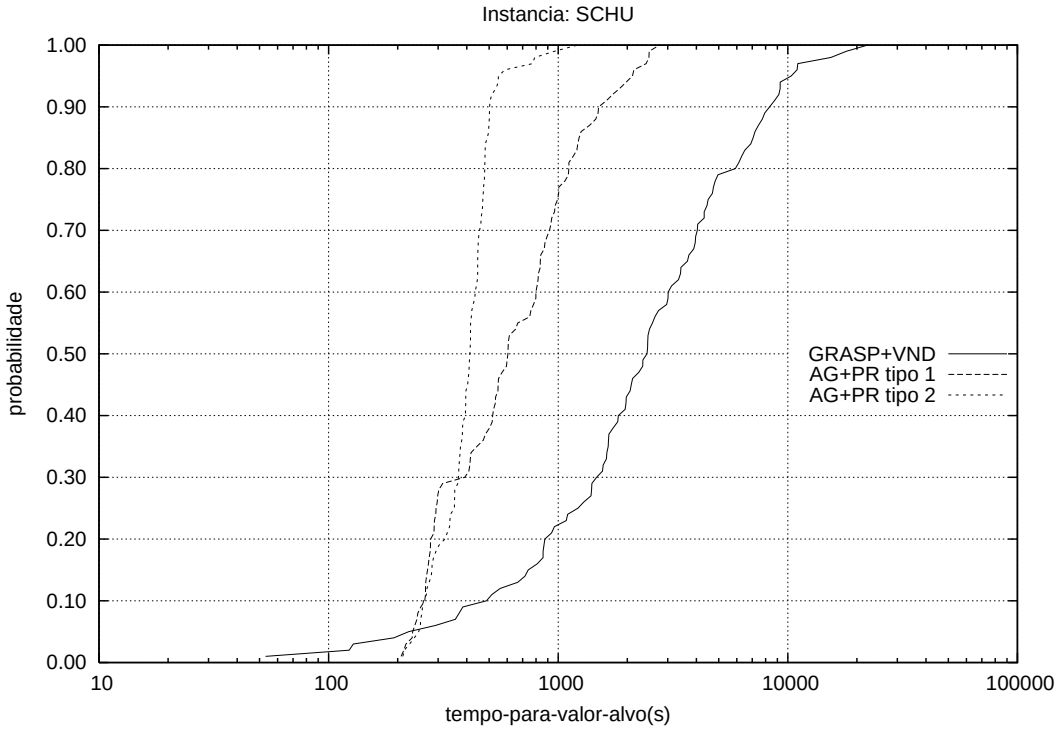

Figura 4.11: Distribuições empíricas de probabilidade do tempo-para-valor-alvo para a instância SCHU para os algoritmos `GRASP+VND`, `AG+PR1` e `AG+PR2`.

O gráfico da Figura 4.11 mostra que as novas heurísticas `AG+PR1` e `AG+PR2` são capazes de encontrar soluções com o mesmo valor daquelas encontradas pelo algoritmo `GRASP+VND` em um tempo computacional muito menor. As novas heurísticas são mais robustas, sendo que a heurística `AG+PR2` foi a que mais se destacou.

Outras duas informações importantes do algoritmo `AG+PR2` foram extraídas deste experimento. Na primeira, foi verificado quantas vezes o procedimento de renovação da população descrito na Seção 4.4 foi realizado em

cada execução. Foi verificado que em 61% das execuções não foi necessário realizar o procedimento de renovação da população para obter uma solução pelo menos tão boa quanto o alvo; em 30% este procedimento foi realizado uma única vez; e em 9% este procedimento foi realizado entre duas e quatro vezes. Esses resultados mostram que o algoritmo `AG+PR2` é bastante eficiente pois na maioria das execuções este atingiu o valor alvo (antigo melhor valor da literatura, portanto um alvo muito difícil) sem precisar realizar a renovação da população.

A segunda informação adicional foi verificar em que etapa do algoritmo `AG+PR2` o alvo foi atingido em cada execução. Foi verificado que o alvo nunca foi atingido por uma solução obtida diretamente pelo algoritmo `GStep_wR`, ou seja, nas etapas de definição da população inicial e reconstrução da classe C durante a fase de evolução da população. Em todas as execuções o alvo foi atingido por uma solução proveniente da etapa de cruzamento, sendo que em 33% o alvo foi atingido pela reconexão por caminhos e em 67% o alvo foi atingido depois de uma reconexão por caminhos seguida de uma operação de busca local. Esses resultados mostram a eficiência do operador genético de cruzamento proposto.

No segundo experimento, apenas foram testados os algoritmos `GRASP+VND` e `AG+PR2`. Estes testes foram realizados sobre as oito intâncias testes da literatura apresentadas na Tabela 2.1, e para as vinte instâncias geradas aleatoriamente descritas na Tabela 3.1. Neste experimento, foi dado o mesmo tempo computacional para cada algoritmo. Dez execuções de 1000 segundos cada foram realizadas para cada instância. Os resultados computacionais para as instâncias testes da literatura estão descritos na Tabela 4.12, e para as vinte instâncias testes geradas aleatoriamente estão descritos na Tabela 4.13. Para cada instância são apresentados os valores médios obtidos, além da melhor solução encontrada após dez execuções de cada algoritmo. Foi indicado em negrito quando um dos algoritmos encontrou estritamente melhores resultados do que o outro. A nova heurística obteve melhores médias de soluções para cinco das oito instâncias da literatura e para todas as instâncias geradas aleatoriamente. A heurística `AG+PR` também encontrou melhores soluções para duas das oito instâncias da literatura, e para todas, exceto uma, das instâncias geradas aleatoriamente.

Baseado nos resultados computacionais obtidos, duas novas implementações foram realizadas. A primeira combina o algoritmo `GRASP+VND` com a reconexão por caminho do tipo 2, gerando o algoritmo `GRASP+VND+PR2`. A Figura 4.14 apresenta o algoritmo desta nova estratégia, que é uma extensão do algoritmo `GRASP+VND` apresentado na

	Média das soluções obtidas (após dez execuções)		Melhor solução obtida (após dez execuções)	
Instância	GRASP+VND	AG+PR	GRASP+VND	AG+PR
GRIS	172.0	172.0	172	172
ANGI	216.0	216.0	216	216
TENU	682.0	682.0	682	682
ETHE	372.8	**372.4**	372	372
ROPA	326.0	**325.8**	325	325
GOLO	498.2	**496.2**	497	**496**
SCHU	761.0	**759.2**	759	759
CARP	552.4	**548.6**	550	**548**

Figura 4.12: Resultados comparativos para dez execuções dos algoritmos GRASP+VND e AG+PR, para as oito instâncias testes da literatura.

Figura 3.11, onde um *pool* de soluções é mantido e a cada $FreqPR = 2$ iterações a solução obtida pelo algoritmo GRASP é submetida a um refinamento via reconexão de caminhos do tipo 2. Este refinamento não foi aplicado a cada iteração, pois foi verificado que isto exigiria um tempo computacional bem maior quando comparado com a versão que o aplica a cada duas iterações. Na linha 11 verifica-se se a solução refinada s' é melhor que a pior solução existente no *pool* e também se não existe uma solução igual a s' no *pool*. Se esses dois critérios forem satisfeitos, na linha 12 a solução s' é inserida no *pool*. Nas linhas 15 e 16 escolhe-se aleatoriamente uma solução s'' do *pool* e realiza-se uma reconexão por caminhos do tipo 2 entre s' e s''. Na linha 22 verifica-se se a solução s''', encontrada após a reconexão por caminhos, é melhor que a pior solução existente no *pool* e também se não existe uma solução igual a s''' no *pool*. Se esses dois critérios forem satisfeitos, na linha 23 a solução s''' é inserida no *pool*.

A outra implementação, denominada AG+PR1E2, utiliza ambas as estratégias de reconexão por caminhos. A estratégia do tipo 1 é aplicada em 25% das soluções, enquanto a estratégia do tipo 2 é aplicada no restante, 75% das soluções. Nesta estratégia a freqüência da busca local foi de $freqBuscaLocal = 7$ gerações. O valor usado para $freqBuscaLocal$ foi o mesmo que no algoritmo genético utilizando apenas o religamento de caminhos do tipo 2 pois a população nestes dois algoritmos converge aproximadamente com a mesma velocidade.

Foi refeito o experimento tempo-para-valor-alvo, onde cem execuções dos algoritmos GRASP+VND, AG+PR1 e AG+PR2 foram realizadas para a instância SCHU e uma execução só terminava quando uma solução com valor de parcimônia menor ou igual a 760 (valor alvo) fosse encontrada. Este

	Média das soluções obtidas (após dez execuções)		Melhor solução obtida (após dez execuções)	
Instância	GRASP+VND	AG+PR	GRASP+VND	AG+PR
TST01	550.8	**549.6**	549	549
TST02	1367.2	**1363.6**	1361	**1358**
TST03	843.8	**840.6**	843	**838**
TST04	599.2	**595.0**	598	**592**
TST05	796.8	**794.0**	793	**790**
TST06	606.6	**605.4**	605	**603**
TST07	1288.6	**1280.6**	1283	**1276**
TST08	873.0	**867.4**	868	**863**
TST09	1159.2	**1154.2**	1156	**1150**
TST10	732.0	**728.6**	730	**725**
TST11	554.8	**546.8**	554	**544**
TST12	1242.0	**1233.0**	1237	**1229**
TST13	1532.4	**1530.6**	1529	**1526**
TST14	1182.0	**1177.4**	1180	**1174**
TST15	774.8	**766.4**	769	**765**
TST16	550.6	**547.6**	547	**545**
TST17	2479.0	**2470.8**	2475	**2468**
TST18	1554.6	**1548.2**	1548	**1542**
TST19	1041.0	**1033.0**	1035	**1028**
TST20	685.4	**678.8**	682	**676**

Figura 4.13: Resultados comparativos para dez execuções dos algoritmos GRASP+VND e AG+PR, para as vinte instâncias testes geradas aleatoriamente.

novo teste envolve o algoritmo AG+PR2 e as duas novas implementações propostas (GRASP+VND+PR2 e AG+PR1E2). As distribuições empíricas de probabilidade para a variável tempo-para-valor-alvo são traçadas na Figura 4.15.

O gráfico da Figura 4.15 mostra que o algoritmo GRASP+VND+PR2 não obteve bons resultados quando comparado com os outros dois algoritmos. Já o algoritmo AG+PR1E2 se mostrou, neste teste, ligeiramente superior ao algoritmo AG+PR2. Este resultado motivou a realização de mais um experimento entre os algoritmos AG+PR2 e AG+PR1E2. Este experimento foi realizados sobre as oito intâncias testes da literatura apresentadas na Tabela 2.1, e para as vinte instâncias geradas aleatoriamente descritas na Tabela 3.1. Neste experimento, foi dado o mesmo tempo computacional para cada algoritmo. Dez execuções de 1000 segundos cada foram realizadas para cada instância. Os resultados computacionais para as instâncias testes da literatura estão descritos na Tabela 4.16, e para as vinte instâncias testes geradas aleatoriamente estão descritos na Tabela 4.17. Para cada instância são apresentados os valores médios obtidos, além da melhor

solução encontrada após dez execuções de cada algoritmo. Foi indicado em negrito quando um dos algoritmos encontrou resultados estritamente melhores do que o outro. O algoritmo AG+PR1E2 obteve melhores médias de soluções para três das oito instâncias da literatura e para quatorze das vinte instâncias geradas aleatoriamente. Ele só foi superado em quatro das vinte e oito instâncias testadas. O algoritmo AG+PR1E2 também encontrou melhores soluções para oito das vinte instâncias geradas aleatoriamente. Ele só foi superado em quatro das vinte instâncias geradas aleatoriamente.

Estes resultados mostram que o algoritmo AG+PR1E2, que combina os dois tipos de religamento de caminhos, foi ligeiramente superior, tanto em tempo computacional quanto em qualidade das soluções obtidas, ao algoritmo AG+PR2, que era o melhor algoritmo genético até o momento. Como mostrado na Figura 4.11 e nas Tabelas 4.12 e 4.13, o algoritmo AG+PR2 superou, tanto em tempo computacional quanto em qualidade das soluções obtidas, o algoritmo GRASP+VND descrito no Capítulo 3 que é o melhor algoritmo GRASP desenvolvido para o problema da filogenia. Assim, o algoritmo AG+PR1E2 é o melhor algoritmo desenvolvido neste trabalho para o problema da filogenia.

4.7 Conclusão

Neste capítulo foram implementados dois algoritmos, AG+PR1 e AG+PR2. Ambos são algoritmos genéticos cuja etapa de cruzamento de indivíduos é realizada por reconexão por caminhos. O que diferencia os dois algoritmos é a maneira pela qual uma folha mal posicionada na solução inicial é selecionada para remoção e o critério utilizado para escolher em que aresta ela deve ser reinserida. Ambos algoritmos foram comparados com o algoritmo GRASP+VND [65] descrito no Capítulo 3. Os três algoritmos foram submetidos a experimentos utilizando 8 instâncias da literatura e 20 instâncias geradas aleatoriamente. Os testes mostraram que o algoritmo AG+PR2 é superior tanto em qualidade das soluções obtidas quanto em tempo computacional, quando comparado com o algoritmo GRASP+VND. Os testes também mostraram que se os três algoritmos forem executados durante um dado tempo, os algoritmos AG+PR1 e AG+PR2 possuem uma probabilidade maior de encontrar uma solução pelo menos tão boa quanto um determinado alvo, sendo a probabiliade do algoritmo AG+PR2 maior do que a do algoritmo AG+PR1. Os resultados apresentados também mostram a eficiência do operador genético baseado na reconexão por caminhos do tipo

2. Eles mostram que em 33% das execuções, o valor alvo é atingido durante a fase de reconexão por caminhos e em 67% o alvo foi atingido após uma reconexão por caminhos seguida de uma busca local com a vizinhança SPR.

A utilização de reconexão por caminhos como operador de cruzamento é uma contribuição original desta tese.

Também foram mostrados outros dois algoritmos, `GRASP+VND+PR2` e `AG+PR1E2`. O primeiro combina o algoritmo `GRASP+VND` com uma etapa de intensificação da busca via reconexão por caminhos do tipo 2. O segundo é um algoritmo genético que utiliza ambos os tipos de reconexão por caminhos. Este algoritmo se mostrou muito eficiente, superando o algoritmo `AG+PR2`, que era o melhor algoritmo até então.

```
Procedimento GRASP+VND+PR2(maxIterações, semente)
Entrada
    maxIterações - número de iterações a serem executadas;
    semente - semente para geração dos números aleatórios;
Saída
    s* - melhor solução encontrada;
Início
01.     f* ← ∞;
02.     Para k=1, ..., maxIterações faça
03.     Início
04.         s ← GStep_wR(semente);
05.         s' ← VNDFilogenia(s);
06.         Se f(s') < f* então
07.         Início
08.             f* ← f(s');
09.             s* ← s';
10.         Fim-se
11.         Se s' é melhor que a pior solução existente no pool
            e não existe uma solução igual a s' no pool então
12.             Insira s' no pool de soluções;
13.         Se k é multiplo de FreqPR então
14.         Início
15.             Escolha aleatoriamente uma solução s'' do pool;
16.             Aplique a reconexão por caminhos tipo 2 usando as soluções s' e s'',
                gerando a solução s''';
17.             Se f(s''') < f* então
18.             Início
19.                 f* ← f(s''');
20.                 s* ← s''';
21.             Fim-se
22.             Se s''' é melhor que a pior solução existente no pool
                e não existe uma solução igual a s''' no pool então
23.                 Insira s''' no pool de soluções;
24.         Fim-se
25.     Fim-Para
26.     Retorne s*;
Fim-GRASP+VND+PR2
```

Figura 4.14: Pseudo-código da heurística GRASP+VND+PR2.

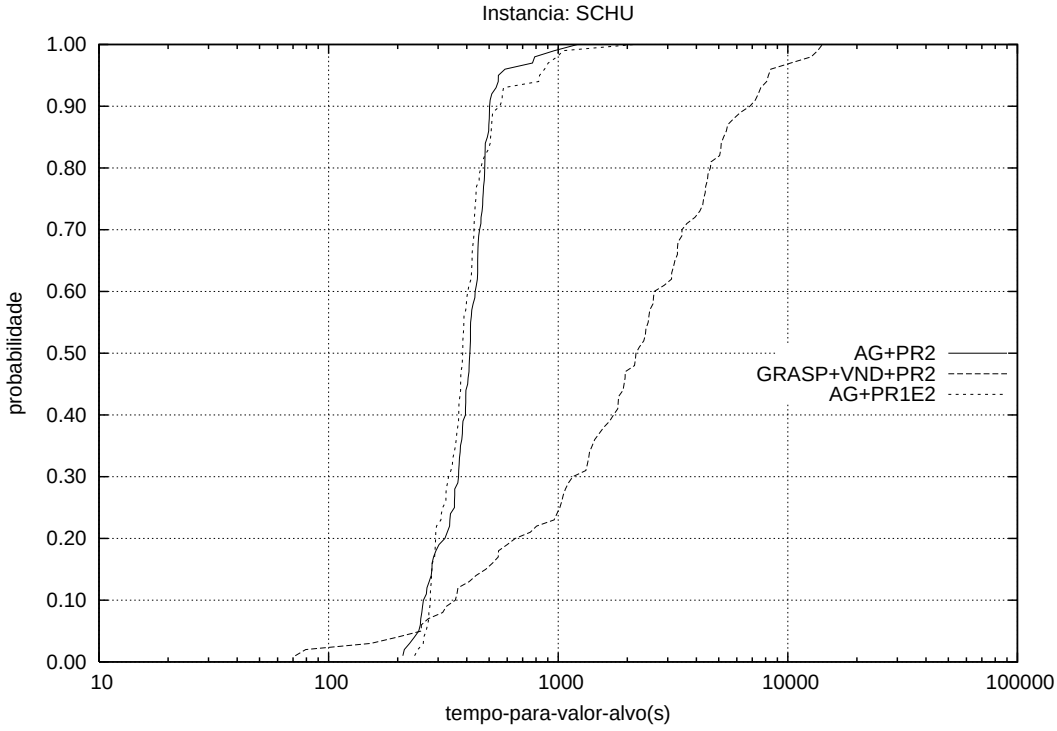

Figura 4.15: Distribuições empíricas de probabilidade do tempo-para-valor-alvo para a instância SCHU para os algoritmos `AG+PR2`, `GRASP+VND+PR2` e `AG+PR1E2`.

	Média das soluções obtidas (após dez execuções)		Melhor solução obtida (após dez execuções)	
Instância	`AG+PR2`	`AG+PR1E2`	`AG+PR2`	`AG+PR1E2`
GRIS	172.0	172.0	172	172
ANGI	216.0	216.0	216	216
TENU	682.0	682.0	682	682
ETHE	372.4	**372.0**	372	372
ROPA	325.8	**325.6**	325	325
GOLO	496.2	496.2	496	496
SCHU	759.2	**759.0**	759	759
CARP	548.6	548.6	548	548

Figura 4.16: Resultados comparativos para dez execuções dos algoritmos `AG+PR2` e `AG+PR1E2`, para as oito instâncias testes da literatura.

	Média das soluções obtidas (após dez execuções)		Melhor solução obtida (após dez execuções)	
Instância	AG+PR2	AG+PR1E2	AG+PR2	AG+PR1E2
TST01	549.6	**549.2**	549	**547**
TST02	1363.6	**1362.0**	1358	1358
TST03	840.6	**839.6**	838	**837**
TST04	595.0	**592.2**	592	**590**
TST05	**794.0**	794.8	**790**	792
TST06	605.4	**605.0**	603	603
TST07	**1280.6**	1282.6	**1276**	1280
TST08	**867.4**	870.0	**863**	866
TST09	1154.2	**1153.6**	1150	1150
TST10	728.6	**727.0**	725	725
TST11	**546.8**	547.4	**544**	545
TST12	1233.0	**1231.4**	1229	1229
TST13	1530.6	**1529.4**	1526	1526
TST14	1177.4	**1175.8**	1174	**1173**
TST15	766.4	**766.0**	765	765
TST16	547.6	**541.8**	545	**538**
TST17	**2470.8**	2472.6	2468	**2464**
TST18	1548.2	1548.2	1542	**1539**
TST19	1033.0	**1032.8**	1028	1028
TST20	678.8	**677.6**	676	**673**

Figura 4.17: Resultados comparativos para dez execuções dos algoritmos AG+PR2 e AG+PR1E2, para as vinte instâncias testes geradas aleatoriamente.