Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estratégia Paralela para Alinhamento Múltiplo de Sequências com Algoritmo Genético Multi-ilha

Lídia Araujo Miranda

Dissertação apresentada como requisito parcial
para obtenção do grau de Mestre em Informática

Orientadora
Profª. Drª. Alba Cristina M. A. de Melo

Coorientador
Prof. Dr. Jan M. Correa

Brasília
2009

Universidade de Brasília – UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Coordenador: Prof. Dr. Li Weigang

Banca examinadora composta por:

Profª. Drª. Alba Cristina M. A. de Melo (Orientadora) – CIC/UnB
Prof. Dr. Mario Antonio Ribeiro Dantas – CTC/UFSC
Prof. Dr. Li Weigang – CIC/UnB

CIP – Catalogação Internacional na Publicação

Lídia Araujo Miranda.

Estratégia Paralela para Alinhamento Múltiplo de Sequências com Algoritmo Genético Multi-ilha/ Lídia Araujo Miranda. Brasília : UnB, 2009. 97 p. : il. ; 29,5 cm.

Tese (Mestre) – Universidade de Brasília, Brasília, 2009.

1. Alinhamente Múltiplo de Sequências, 2. Algoritmo Genético, 3. Algoritmo Paralelo, 4. Injeção de Ilhas

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro – Asa Norte
CEP 70910–900
Brasília – DF – Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estratégia Paralela para Alinhamento Múltiplo de Sequências com Algoritmo Genético Multi-ilha

Lídia Araujo Miranda

Monografia apresentada como requisito parcial
para conclusão do Mestrado em Informática

Profª Drª Alba Cristina M. A. de Melo (Orientadora)
CIC/UnB

Prof. Dr. Mario Antonio Ribeiro Dantas
CTC/UFSC

Prof. Dr. Li Weigang
CIC/UnB

Prof. Dr. Li Weigang
Coordenador do Mestrado em Informática

Brasília, 4 de setembro de 2009

Dedicatória

Dedico essa dissertação a Deus, que a cada dia me dá a chance de recomeçar.

Agradecimentos

Sou muito grata à minha orientadora Alba, pois com doçura e paciência me ajudou a superar as dificuldades.

Agradeço ao meu amigo Caetano, que me ajudou imensamente e todo o trabalho.

Agradeço a minha prima Camila que me ajudou na área de Biologia e também me deu todo apoio.

Agradeço a minha família que esteve comigo em toda minha jornada.

Agradeço aos amigos pelo apoio e motivação.

Resumo

O Alinhamento Múltiplo de Sequências genéticas (AMS) é executado milhares de vezes ao dia por cientistas, a fim de identificar regiões de semelhança entre três ou mais sequências. Os alinhamentos múltiplos assim obtidos são usados na resolução de problemas complexos, como a determinação do histórico evolutivo das espécies. Por se tratar de um problema NP-completo, geralmente são utilizadas soluções heurísticas para a sua resolução. Dentre soluções adotadas, destaca-se o Algoritmo Genético (AG), que é um método iterativo não-determinístico, baseado nos princípios da Evolução das Espécies de Darwin. Apesar de apresentar soluções boas para o AMS, os algoritmos genéticos demandam um alto poder de processamento, que se traduz em um alto tempo de execução. Por essa razão, algumas estratégias paralelas foram propostas na literatura para acelerar a obtenção de alinhamentos múltiplos com AGs, geralmente utilizando a estratégia da ilha como base de paralelização. A presente dissertação de mestrado propõe e avalia uma estratégia paralela que utiliza Algoritmo Genético para o Alinhamento Múltiplo de Sequências, inspirada no modelo Multi-ilha. De maneira diferente das abordagens para AMS existentes na literatura, a estratégia proposta utiliza 3 Super Ilhas, onde cada Super Ilha implementa um modelo tradicional de ilhas. Os resultados obtidos com bases reais de proteínas mostram que a estratégia proposta é capaz de encontrar alinhamentos múltiplos de melhor qualidade em menor tempo, quando comparada com a estratégia de ilha tradicional.

Palavras-chave: Alinhamente Múltiplo de Sequências, Algoritmo Genético, Algoritmo Paralelo, Injeção de Ilhas

Abstract

The Multiple Sequence Alignment (MSA) between genetic sequences is exhaustively done by scientists trying to identify matching regions within three or more sequences. The resulting multiple alignments are used in complex problems like the one of establishing genetic relationships between biological sequences. The MSA has been shown to be an NP-complete problem, therefore heuristic solutions are usually used to solve it. One of the solutions that has shown good results for MSA is the Genetic Algorithm (GA), a non deterministic iterative method, based on Charles Darwin's theory of evolution. Though presenting good results, the GA demands high amount of computing power, taking usually a lot of time to be executed. To speed up the sequential algorithms execution, parallel algorithms were proposed in the literature, most of them using the island strategy of parallelization. This masters dissertation proposes and evaluates a parallel strategy that uses Genetic Algorithms to the Multiple Sequence Alignment based on the Multi-island parallelization strategy. Differently from other MSA strategies, the proposed strategy creates three Super Islands and each one executes a GA parallelized by the island strategy. The results were obtained with real protein banks and revealed that the proposed strategy is capable of finding better multiple alignments in a smaller amount of time, when compared to the conventional island strategy.

Keywords: Multiple Sequence Alignment, Genetic Algorithm, Parallel Algorithm, Island Injection

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

Os Algoritmos Genéticos (AG) [37, 27] são meta-heurísticas que tiveram o desenho de seu modelo inspirado na teoria da evolução de Darwin [16]. O algoritmo aplica a uma população inicial, que é composta de um conjunto de soluções possíveis para o problema, operadores de mutação e de *crossover*, formando um novo grupo de possíveis soluções. Os operadores buscam, a partir da alteração e cruzamento de elementos presentes na população, gerar novos elementos que podem ser melhores que os presentes na população atual. Por meio do operador de seleção, o AG seleciona as soluções que comporão a nova população de forma que a população gerada contenha soluções melhores que a anterior. Esse processo é repetido várias vezes, simulando o processo de evolução dos seres vivos.

A fim de acelerar a obtenção de resultados, foram propostos AGs paralelos. Para a solução de problemas que possuem um cálculo de avaliação de soluções que demandam uma grande quantidade de processamento, implicando em um grande gasto de tempo nessa etapa, geralmente é implementada a estratégia de paralelização mestre/escravo [12, 13]. Para soluções onde os ciclos de geração da população são rápidos, a estratégia mais recomendada é o modelo de ilha (seção 2.6) [12, 13]. A estratégia de ilha executa vários AGs em nós separados e, em intervalos pré-definidos, esses AGs trocam elementos entre si. A estratégia de ilha é o modelo mais popular de AG paralelo [12] e em várias situações, a estratégia de ilha, além de obter resultados bons em um menor número de gerações, obtém resultados melhores que a estratégia sequencial. Em [31], de acordo com o modelo abstrato construído pelos autores, é conjecturado que a estratégia de ilha obtém indivíduos melhores que a estratégia sequencial quando o algoritmo funciona bem para populações pequenas.

Em [28], foi proposta uma nova estratégia para paralelização de AG que utilizava conjuntamente as estratégias de ilha e mestre/escravo. Esse modelo foi chamado de Multi-ilha ou Injeção de Ilhas, propondo a criação de três ilhas que trocam soluções entre si em intervalos determinados. Dentro de cada ilha, é implementado um AG mestre/escravo. Em uma das ilhas, também chamada de Ilha de Alta Resolução, o cálculo de avaliação implementado é um cálculo pesado que obtém soluções muito boas em um tempo longo. Por outro lado, nas outras

duas ilhas (Ilhas de Baixa Resolução), o cálculo é mais leve e rápido. No entanto, as soluções não são tão boas. Segundo os autores, essa estratégia direciona o algoritmo para soluções melhores em um tempo mais curto [28].

Os AGs têm sido amplamente estudados e utilizados com sucesso na resolução de uma variedade de problemas, tais como Gerência de Recursos Hídricos [28], Escalonamento de Atividades diversas [59, 10, 66] e no Alinhamento Múltiplo de Sequências [33, 43].

O Alinhamento Múltiplo de Sequências biológicas (MSA) é o alinhamento de três ou mais sequências de DNA, RNA ou proteínas. Normalmente, as sequências são alinhadas para encontrar alguma relação evolutiva entre elas, seja um ancestral comum ou uma região de código genético que desempenha uma função comum. Alinhamentos bem feitos de sequências são de grande interesse dos cientistas e, por isso, esse processo usualmente é realizado milhares de vezes ao dia em todo mundo. Por ser um problema NP-completo [40], heurísticas são normalmente utilizadas para solucionar o problema MSA. As abordagens heurísticas são classificadas em categorias que variam de acordo com os autores. No entanto, duas permanecem em comum, que são os métodos progressivos e os iterativos. Os métodos progressivos alinham todos os pares de sequências, em seguida, constroem uma árvore filogenética baseada nos alinhamentos obtidos anteriormente. Finalmente, utilizam a árvore filogenética construída para guiar o alinhamento das sequências. Os métodos iterativos produzem uma solução inicial e a refinam repetidas vezes até que não possa mais ser melhorada. Dentre os principais métodos progressivos estão o CLUSTAL-W [26] e o T-COFFEE [11]; e dentre os iterativos estão os Modelos Escondidos de Markov (*Hidden Markov Models* - HMM) [21, 39] e Algoritmos Genéticos [27].

Em [33], o autor mostra que o SAGA é um Algoritmo Genético capaz de obter soluções muito boas, melhores que o CLUSTAL-W, em um tempo razoável. O Algoritmo Genético Quântico descrito em [49], apresenta resultados ainda melhores. Isso mostra que o AG é uma estratégia adequada para a resolução do MSA. Dentre as estratégias paralelas para MSA estudadas na literatura (iPGA [58, 57] e PHGA [64, 65]) nenhuma utilizava a estratégia de Injeção de Ilhas descrita em [28].

1.2 Principais Contribuições

O objetivo da presente Dissertação de Mestrado é propor e avaliar uma estratégia paralela de Injeção de Ilhas para o Problema do Alinhamento Múltiplo de Sequências. Dentre as principais contribuições, podemos citar:

- **Projeto da Organização Multi-ilha para MSA:** diferente de [28], a estratégia proposta nesse trabalho utiliza como estratégia de paralelização dentro das ilhas, também a estratégia de ilha e não a estratégia mestre/escravo. Essa estratégia foi considerada mais adequada, pois não se pretendia executar nenhuma etapa muito demorada dentro do ciclo de evolução da população. Na Ilha de Alta Resolução, foi utilizado o processo convencional de paralelização de ilhas, onde a população é dividida em grupos de

alinhamentos que são enviados às ilhas, a ilha processa o grupo de alinhamento e troca elementos com as outras ilhas em intervalos pré-definidos. Na Ilha de Baixa Resolução, os alinhamentos são divididos em partes e cada ilha é responsável por processar um grupo de partes. Dessa forma, as ilhas dentro das Ilhas de Baixa Resolução estariam processando elementos menores e obtendo soluções de forma mais rápida. Logo, a diferença entre Ilha de Alta Resolução e as de Baixa Resolução não está no cálculo ou no modo como o processamento é feito, e sim no tamanho dos indivíduos que estão sendo processados. Devido à nova proposta de paralelismo feita para a Ilha de Baixa Resolução (evolução de partes dos alinhamentos), foram propostas uma nova estratégia de migração e novos operadores.

As ilhas dentro da Ilha de Baixa Resolução não podem trocar elementos entre si, pois os elementos de cada ilha correspondem a um grupo de partes específicas, sendo que na ilha 1 está o grupo de primeiras partes dos alinhamentos, na ilha 2 está o grupo de segundas partes e assim por diante. Então, propusemos que essa troca de elementos seja feita da seguinte forma. Primeiramente, todos os grupos de partes são enviados a um nó central pelas ilhas. Nele, são executados três operadores: um operador que junta as partes formando um novo grupo de alinhamentos; um operador de mutação e um operador de corte. Depois de executados os operadores, os grupos de partes são enviados novamente para cada ilha.

O operador de junção de partes, proposto nesse trabalho, reúne as mesmas partes que tinham sido separadas anteriormente em um novo alinhamento.

O operador de corte, também proposto nessa dissertação, parte os alinhamentos de forma que todos os elementos presentes em um grupo de partes possuam os mesmos caracteres. Os cortes são feitos em posições diferentes do alinhamento a cada troca de elementos entre as ilhas, de maneira a reduzir a probabilidade de convergência prematura.

- **Operadores de Mutação com cálculo SP integrado:** dois novos operadores de mutação foram propostos. Os operadores movem caracteres de uma coluna para outra sobre subsequências formadas por espaços, com o objetivo de formar alinhamentos mais aptos. O cálculo SP é utilizado para garantir que o operador encontre sempre uma solução mais apta ou tão apta quanto a anterior.

1.3 Estrutura do Documento

O presente documento está organizado da seguinte maneira. O Capítulo 2 discorre sobre os conceitos base de Algoritmos Genéticos (AG), tais como: comportamento padrão de um Algoritmo Genético típico, os principais operadores e estratégias paralelas para AG. Em seguida, no Capítulo 3, é abordado o Problema do Alinhamento Múltiplo de Sequências e as principais abordagens para solucionar o mesmo são apresentadas. Depois, no Capítulo 4, são descritos quatro Algoritmos Genéticos para o problema do MSA, sendo três deles paralelos e, ao

final, é feita uma análise de cada algoritmo. No capítulo 5, é descrita a estratégia de Algoritmo Genético Multi-ilha proposta nesse trabalho. Então, no Capítulo 6, são apresentados e analisados os resultados dos testes feitos com o algoritmo proposto. Finalmente, o capítulo 7 apresenta a conclusão e os trabalhos futuros.

Capítulo 2

Algoritmos Genéticos

Os Algorimos Genéticos (AG) são algoritmos não lineares e estocásticos (não determinísticos) de otimização e busca [38] que têm seu funcionamento baseado no princípio da seleção natural de Charles Darwin [16]. Foram introduzidos por John Holland [37] e popularizados por um de seus alunos, David Goldberg [27]. Baseado no princípio da seleção natural, o algoritmo simula a formação de gerações de soluções, onde uma geração é formada a partir da geração anterior aplicando-se regras de seleção das soluções mais fortes, mutações, cruzamento de soluções ou outras manipulações que porventura possam ter sido definidas, de modo que cada nova geração seja melhor que a anterior, isto é, mais evoluída.

O método de otimização e busca consiste em tentar várias soluções presentes em um espaço de busca, local onde estão todas as possíveis soluções do problema [14], e a partir do resultado dessas tentativas gerar soluções melhores até atingir um resultado satisfatório. A Figura 2.1 apresenta o Algoritmo Genético Típico [14].

Algoritmo Genético Típico

```
1. Seja S(t) a população de cromossomos na geração t.
2.
3. t ← 0
4. inicializar S(t)
5. avaliar S(t)
6. enquanto o critério de parada não for satisfeito faça
7.         t ← t+ 1
8.         selecionar S(t) a partir de S(t- 1)
9.         aplicar crossover sobre S(t)
10.        aplicar mutação sobre S(t)
11.        avaliar S(t)
12. fim enquanto
```

Figura 2.1: *Algoritmo Genético Típico [14].*

O primeiro passo de um AG é gerar uma população inicial de "cromossomos". Essa população representa um conjunto de soluções possíveis para o problema

(linha 4). Depois de gerada, a população é avaliada e a cada solução é atribuída uma pontuação que representa a qualidade da mesma (linha 5). Essa pontuação é chamada de aptidão. Aplicando um conjunto de regras predefinidas na população inicial, o algoritmo gera uma segunda população que tem o objetivo de ser mais apta que a primeira. As regras utilizadas para gerar uma nova população no Algoritmo Genético Típico (Figura 2.1) são seleção (linha 8), *crossover* (linha 9) e mutação (linha 10). Depois de gerada a nova população, ela é avaliada. Se a população for considerada boa o suficiente, o algoritmo pára, e essa população será o resultado do algoritmo. Senão, novas populações são geradas até que o critério de parada seja atingido (linha 6).

Geralmente o problema da otimização consiste em encontrar o ponto de máximo ou o ponto de mínimo de uma função $f(x_1, .., x_n)$, chamada de função objetivo. Como ilustrado na Figura 2.2, os pontos de máximo são os pontos do domínio onde a imagem é maior que a imagem dos elementos próximos a eles. Os pontos que possuírem imagem maior que todos os pontos de máximo são denominados máximos globais, enquanto os outros são chamados de máximos locais. O ponto de mínimo segue a mesma filosofia só que, ao invés do ponto possuir a maior imagem, ele possuirá a menor. O Algoritmo Genético, na procura do máximo global, pode confundir-se com os outros pontos de máximo levando o algoritmo a apontar um ponto de máximo local ao invés do máximo global. Esse problema é chamado da convergência prematura [47]. Os máximos e mínimos locais, também podem ser chamados de ótimos locais.

Os elementos a serem processados pelo AG são os cromossomos que representam uma possível solução para o problema. Em geral, o cromossomo representa o conjunto de parâmetros de uma função e as combinações de todos as configurações de parâmetros possíveis formam o espaço de busca.

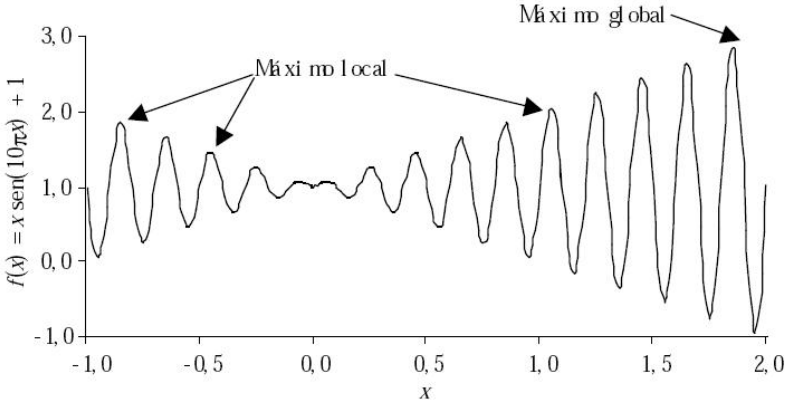

Figura 2.2: *Pontos de máximo local e global [14].*

2.1 População Inicial

Como já mencionado anteriormente, a geração da população inicial é feita na primeira etapa do algoritmo. Essa população corresponde a um conjunto de soluções possíveis para o problema, sendo gerada a partir do conjunto total de soluções possíveis [14]. Não existe uma forma padrão para gerar a população inicial de um Algoritmo Genético, podendo ser obtida a partir da aplicação de qualquer regra. No entanto, como todas as outras populações geradas terão sua raiz na população inicial, é interessante pensar que característica(s) seria(m) importante(s) que a população inicial tivesse de modo a evitar falhas que repercutam nas gerações seguintes.

A população inicial é uma amostra do conjunto de todas as soluções possíveis, devendo essa refletir o conjunto como um todo. Caso contrário, pode ser que alguma característica importante seja perdida nessa estapa. Uma forma de evitar que perdas ocorram é gerar a população inicial de maneira uniforme, procurando dentro do espaço de busca selecionar soluções com intervalos uniformes entre si. Outras alternativas podem ser construídas mesclando esse método com outros. Um exemplo seria a selecionar metade da população de forma aleatória e a outra metade de forma uniforme.

Quando o espaço de busca é pequeno, esse está mais propício à perda de características na montagem da população inicial, então é interessante que a população incial seja maior que as populações de gerações subsequentes de forma a minimizar o risco de perda.

Também existe um método denominado *seeding* que consiste na inserção na população inicial de soluções encontradas por outros métodos de otimização, de modo a direcionar o algoritmo para os pontos ótimos e assim obter melhores resultados.

2.2 Avaliação

A avaliação é uma etapa do algoritmo que avalia uma população dada como entrada, pontuando cada solução de acordo com sua aptidão. As soluções consideradas mais próximas do resultado que se procura devem receber um valor maior de aptidão, de modo que as próximas etapas do algoritmo utilizem essa informação e gerem uma nova população que seja mais apta que a primeira.

2.3 Seleção

A seleção escolhe geralmente os melhores cromossomos da população atual e os coloca em uma população intermediária que será utilizada para gerar os filhos da próxima geração por meio da aplicação dos operadores de *crossover* e mutação. O processo de seleção se assemelha ao processo de seleção natural, onde os mais aptos tem a reprodução bem sucedida e consequentemente suas características

serão repassadas para a próxima geração. Geralmente, os pais são selecionados de maneira proporcional à sua aptidão.

Roda Roleta: O método da Roda Roleta é um método de seleção bem popular. Primeiramente, o método monta uma coluna de aptidões acumuladas, onde o valor da aptidão acumulada do cromossomo i é a soma da aptidão de i com a soma das aptidões dos $i-1$ cromossomos anteriores a ele. Depois disso, o método gera um valor aleatório no intervalo de 0 (zero) e o valor total de aptidões acumuladas. O cromossomo que possuir a aptidão acumulada logo após do valor aleatório gerado é selecionado para integrar a população intermediária. Esse processo é repetido até que a população intermediária seja preenchida [14]. Caso a função de avaliação gere valores de aptidão negativos, o algoritmo Roda Roleta não pode ser utilizado.

Seleção por Torneio: O método seleciona n cromossomos aleatoriamente com a mesma probabilidade e dentre esses n escolhe o de maior aptidão para integrar a população intermediária, esse processo é repetido até que a população intermediária seja preenchida [14].

2.4 Geração de Filhos

A etapa de geração de filhos utiliza a população intermediária obtida na etapa de seleção para, a partir da aplicação de mecanismos de busca, gerar filhos mais evoluídos que os filhos da geração anterior. Nessa etapa, serão descritos e exemplificados os dois processos mais comuns utilizados para gerar filhos, o *crossover* e a mutação, ambos inspirados no processo evolutivo [16].

A substituição de uma geração antiga por uma nova pode ser feita de diversas maneiras. Duas das formas mais comuns são a substituição dos N pais (geração antiga) pelos N cromossomos filhos (geração nova) e a substituição dos N cromossomos pais pelos N melhores do conjunto de pais e filhos. Nesses dois métodos, uma geração pode variar bastante em relação à precedente, tendendo a variar menos ao passar das gerações. Outro tipo de substituição é o *steady-state* (estado estável), em que somente um ou dois novos filhos são gerados e substituem os dois piores da população anterior. Nessa técnica é interessante evitar filhos duplicados na população.

Como formas de evitar filhos duplicados, temos:

1. Evitar gerar cromossomos idênticos aos da população inicial;
2. Observar se o filho é igual a um dos pais;
3. Verificar se foi aplicado mutação ou *crossover* nos pais, pois se não o filhos será igual a um dos pais;

4. Manter na população todos os cromossomos distintos entre si;

Pode ocorrer também dos filhos gerados não estarem dentro do domínio de soluções possíveis. Nesse caso, o filho é chamado de infactível. Uma abordagem para se lidar com o filho infactível é atribuir uma aptidão 0 (zero) a ele, de forma que ele não deverá ser selecionado para compor a próxima população intermediária. Mas existem casos onde os filhos infactíveis estão próximos de regiões factíveis, o que pode interessar para a realização de *crossovers*, então uma alternativa é apenas o penalizar o valor da aptidão do cromossomo.

Geralmente, para garantir que algumas características permaneçam de uma geração para a outra e não sejam eliminadas na etapa geração de filhos, é utilizado um método chamado elitismo [47]. Nesse método, um ou mais cromossomos, com os melhores valores de aptidão, são repassados para a geração posterior sem sofrer alterações.

2.4.1 Operador de *Crossover*

O *crossover* simula o processo de reprodução dos seres vivos, onde algumas características de cada pai são repassadas para o filho. O *crossover* objetiva que as características mais fortes do pais sejam repassadas para os filhos, de modo que a nova geração esteja mais próxima do resultado ótimo.

A probabilidade do operador de *crossover* ser aplicado a um par de soluções é chamada de taxa de *crossover*.

Operador de *Crossover* Binário

O *crossover* binário é aplicado em cromossomos que são formados por *bits*, conforme ilustrado na Figura 2.3. Os tipos de *crossover* binário mais conhecidos são o de n pontos e o uniforme.

***Crossover* de N pontos**: Aplicando-se o operador de *crossover* de 1 ponto a um par de cromossomos retirados da população intermediária, serão gerados dois cromossomos filhos. O operador corta cada cadeia de bits dos cromossomos pais em uma posição aleatória, produzindo duas partes iniciais e duas partes finais. As partes iniciais dos cromossomos são trocadas, sendo que os cromossomos filhos tem a parte inicial vinda de um pai e a parte final do outro. A Figura 2.3 ilustra o *crossover* de 1 ponto.

$$
\begin{array}{ll}
pai_1 & (\underline{0010101011}|\underline{100000111111}) \\
pai_2 & (0111101000|100110100110) \\
 & \\
filho_1 & (\underline{0010101011}|100110100110) \\
filho_2 & (0111101000|\underline{100000111111})
\end{array}
$$

Figura 2.3: *Crossover 1 ponto [18].*

pai_1 (001010|10111000|00111111)
pai_2 (011110|10001001|10100110)

$filho_1$ (001010|10001001|00111111)
$filho_2$ (011110|10111000|10100110)

Figura 2.4: *Crossover 2 pontos [18].*

O *crossover* de 2 pontos corta os cromossomos pais em 2 pontos, gerando os filhos a partir da troca das partes do meio dos pais, conforme ilustrado na Figura 2.4.

O *crossover* de 4 pontos corta os cromossomos pais em 4 pontos. Os filhos são gerados a partir da troca das segunda e quarta partes dos cromossomos pais, conforme ilustrado na Figura 2.5.

pai_1 (0010|101011|1000001|11|111)
pai_2 (0111|101000|1001101|00|110)

$filho_1$ (0010|101000|1000001|00|111)
$filho_2$ (0111|101011|1001101|11|110)

Figura 2.5: *Crossover 4 pontos [18].*

Esse processo pode ser utilizado para outros valores de n, no entanto o mais comumente usado é para 1, 2 e 4 pontos.

***Crossover* uniforme**: No *crossover* uniforme é gerada uma máscara de bits aleatória que determina de qual pai o filho receberá o bit em dada posição. Por exemplo, se o n-ésimo bit da máscara for 1 o n-ésimo bit do filho receberá o n-ésimo bit do pai_1, caso contrário, o bit virá do pai_2. Conforme ilustrado na Figura 2.6.

Máscara	(	1	0	1	1	0	0	1	1	)
pai_1	(	0	0	1	0	1	0	1	0	)
		↓		↓	↓			↓	↓	
$filho$	(	0	1	1	0	1	0	1	0	)
			↑			↑	↑			
pai_2	(	0	1	1	1	1	0	1	0	)

Figura 2.6: *Crossover uniforme [18].*

Operador de *Crossover* Real

Caso os cromossomos sejam números reais, alguns dos operadores de crossover mais comuns são: *crossover* média [17], *crossover* média geométrica [17], *crossover* BLX-α [56], *crossover* linear [46], *crossover* aritmético [47], *crossover* heurístico [47] e *crossover* simples [47].

2.4.2 Operador de Mutação

O operador mutação altera de forma pontual o valor do cromossomo, destruindo a informação anterior e construindo uma nova. Esse processo diversifica a população dos cromossomos, assim como a mutação no processo evolutivo. Por se tratar de um operador destrutivo ele deve ser aplicado com uma taxa baixa (taxa de mutação) entre 0,1% e 5%, o suficiente para garantir a diversidade [47].

Operador de Mutação Binária

O operador mutação binária altera o valor de 1 ou mais bits do cromossomo, conforme ilustrado na Figura 2.7.

$$pai \quad (\underline{0}010\underline{1}01011100000\underline{1}111\underline{1}1)$$

$$filho \quad (\underline{1}010\underline{0}01011100000\underline{0}111\underline{0}1)$$

Figura 2.7: *Mutação binária [18].*

Operador de Mutação Real

Caso os cromossomos sejam números reais, alguns dos operadores de mutação mais comuns são: mutação uniforme [47], mutação não-uniforme [47] e mutação de limite [47].

2.5 Critério de Parada

O critério de parada de um Algoritmo Genético define em que circunstâncias o algoritmo deve parar de produzir novas gerações. Em geral, são utilizados os seguintes critérios de parada:

1. **Resultado ótimo:** o algoritmo pára quando o resultado ótimo é encontrado.
2. **Número de gerações:** é definido um número finito de gerações que o algoritmo pode gerar. Quando é atingido esse valor o algoritmo pára.
3. **Convergência:** o algoritmo pára quando não ocorrer melhora significativa no cromossomo de maior aptidão em um determinado número de gerações.

Outra alternativa consiste em parar o algoritmo quando 90% à 95% dos cromossomos convergirem para uma mesma região, o que provavelmente indica que a população convergiu para uma solução de maior aptidão, mais evoluída, isto é, mais próximo do melhor resultado. É importante que durante o processo evolutivo o algoritmo utilize técnicas para evitar que os cromossomos convirjam para um ótimo local.

2.6 Algoritmos Genéticos Paralelos

Apesar dos Algoritmos Genéticos serem uma meta-heurística poderosa para a solução de problemas de otimização, seu tempo de execução pode ser bastante considerável. Uma das alternativas para a execução de Algoritmos Genéticos em tempo hábil, é a busca de estratégias paralelas. Essas estratégias são divididas em três categorias de acordo com Erick Cantú-Paz [12, 13]:

1. **População única mestre/escravo (*global single-population master-slave GAs*):** O modelo de mestre/escravo é igual ao modelo de AG sequencial, se diferenciando apenas pelo cálculo da aptidão que é distribuído. O processador mestre armazena a população, executa as operações do AG e distribui os filhos gerados. Os escravos calculam a aptidão dos filhos e retornam os resultados ao mestre que dará continuidade à sua execução. O modelo população única mestre/escravo é ilustrado na Figura 2.8.

2. **População única refinada (*single-population fine-grained*):** No modelo refinado, os nós são organizado em uma matriz de duas dimensões. A população é espacialmente dividida entre os nós que executarão um AG. Cada nó compartilha uma área com os nós vizinhos, onde parte da população é compartilhada. Essa população pode ser utilizada pelos AGs vizinhos. Dessa forma, indiretamente, é possível a comunicação entre todos os nós. O modelo população única refinado é ilustrado na Figura 2.9.

3. **Múltiplas populações (*multiple-population coarse-grained* ou Modelo de ilha):** Nesse modelo, existem várias populações distribuídas em ilhas que estão organizadas em uma topologia de anel. Em cada ilha é executado um AG que, em intervalos pré-definidos (intervalo de migração), trocam alinhamentos com as ilhas vizinhas. Esse método também é conhecido como Algoritmo Genético de ilhas (do inglês *island genetic algorithms*) e é ilustrado na Figura 2.10.

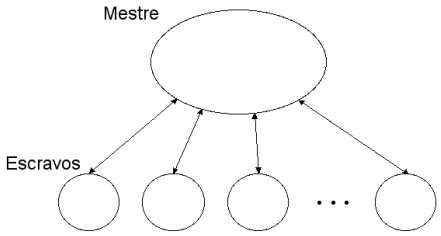

Figura 2.8: *Modelo de AG paralelo população única mestre/escravo [12, 13].*

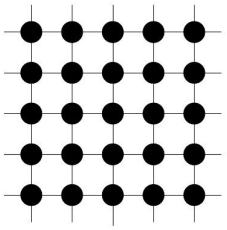

Figura 2.9: *Modelo de AG paralelo população única refinada [12, 13].*

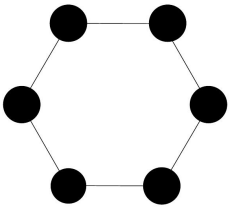

Figura 2.10: *Modelo de AG paralelo de ilha [12, 13].*

2.7 Algoritmo Genético Multi-ilha

Em [28], Babbar et al. propuseram uma estratégia de AG paralelo para melhorar a performance do AGS (Algoritmo Genético Sequencial) na resolução do problema da gerência de recursos hídricos. A estratégia Multi-ilha combina duas estratégias de paralelização: ilha e mestre/escravo (seção 2.6).

A estratégia de ilha é aplicada criando-se três ilhas e em cada uma existe um AG mestre/escravo. Em uma das ilhas, o AG é implementado para obter soluções melhores com o cálculo de aptidão mais demorado (de alta resolução) e nas outras duas o cálculo implementado é mais leve (de baixa resolução). Os AGs mais leves trocam entre si os melhores indivíduos ocasionalmente e enviam os melhores indivíduos para o AG mais pesado.

Na estratégia mestre/escravo, o mestre executa todas as funções de um AG comum, no entanto, no momento do cálculo de aptidão ele envia as soluções para os escravos para serem avaliadas e depois os resultados são retornados ao mestre. Pelo fato do cálculo de aptidão no problema de gerência de recursos hídricos demandar muito processamento, a abordagem mestre/escravo é adequada pois a economia no tempo de computação da aptidão compensa o custo de comunicação [28]. Segundo os autores, essa estratégia direciona o algoritmo para soluções melhores em um tempo mais curto [28]. A Figura 2.11 ilustra o modelo.

Em [28], a estratégia Multi-ilha foi referida como Algoritmo Genético de Injeção de Ilhas (do inglês, *Island Injection Genetic Algorithm*).

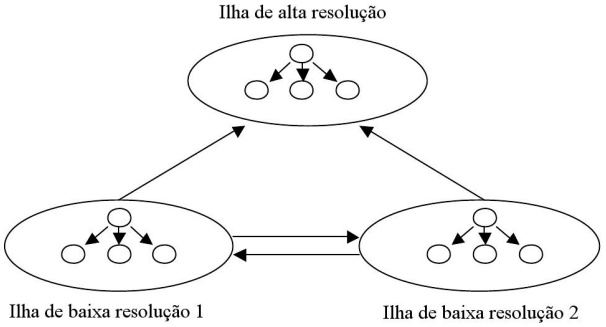

Figura 2.11: *Estratégia paralela descrita em [28].*

Em [31], de acordo com o modelo abstrato construído pelos autores, é conjecturado que a estratégia de ilha obtém indivíduos melhores que a estratégia sequencial quando os algoritmo funciona bem para populações pequenas. Isso pode ser verdadeiro tanto para problemas linearmente separáveis como para problemas que não fazem parte dessa classe.

Capítulo 3

Alinhamento de Sequências Biológicas

3.1 Introdução

O alinhamento de sequências biológicas é uma operação realizada milhares de vezes ao dia, ao redor do mundo, que visa identificar similaridades entre as sequências, de maneira a inferir características das mesmas [50]. O alinhamento organiza as sequências em linhas e colunas, de forma que cada linha representa uma sequência e as regiões associadas estão no mesmo conjunto de colunas. Todas as linhas tem o mesmo comprimento e isso é possível pela eventual inserção de espaços vazios nas sequências. O alinhamento de duas sequências é chamado alinhamento dois-a-dois e o alinhamento de um número maior de sequências é chamado alinhamento múltiplo.

O Alinhamento Múltiplo de Sequências (MSA) é muito utilizado por biólogos que buscam identificar regiões similares em um conjunto de sequências. Por se tratar de um problema NP-completo [40], a solução do MSA de forma determinística consumiria uma quantidade de recursos computacionais que torna inviável a sua execução. Então, métodos heurísticos, que simplificam as restrições do problema, racionalizando o uso de memória e processamento, estão sendo exaustivamente estudados [30].

3.2 Conceitos Básicos

O ácido desoxirribonucléico (DNA) é uma molécula em forma de uma escada em espiral, onde os degraus representam as ligações (pontes) de hidrogênio que conectam dois nucleotídeos. O nucleotídeo é uma subunidade do DNA, composto de uma base nitrogenada, fosfato e açúcar. As combinações das bases (A - adenina, G - guanina, C - citosina e T - timina) que diferenciam um DNA do outro [3].

O ácido ribonucléico (RNA) é uma molécula formada a partir do DNA. É semelhante ao DNA, mas, ao invés, de ser formada por duas tiras conectadas através de ligações de hidrogênio, a molécula do RNA é composta somente por uma tira de bases (A - adenina, G - guanina, C - citosina e U - uracila) [3].

A proteína é uma molécula composta por uma ou mais cadeias de aminoácidos (estrutura base da proteína). Suas características são determinadas pela ordem das bases do RNA a partir do qual ela é sintetizada [3].

No processo evolutivo, é natural que o DNA do ser vivo venha sofrendo algumas mutações e alterações no material genético. As técnicas de alinhamento de sequências simulam essas mutações de modo que o alinhamento final obtido reflita a relação das sequências com um ancestral comum [30]. As mutações geralmente consideradas são [55]:

- **Remoções:** uma ou mais bases são eliminadas de uma sequência.
- **Substituições de bases:** substituição de uma base da sequência por outra base.
- **Inserções:** uma ou mais bases são inseridas em uma sequência.

A forma como essas mutações são simuladas varia de acordo com o algoritmo. Geralmente, os algoritmos atribuem uma pontuação à comparação de cada par de bases. Na comparação do DNA, do RNA e das proteínas, geralmente atribui-se um valor positivo quando os caracteres são iguais e um valor negativo quando os caracteres são diferentes. Quando uma das bases da comparação for vazia, o que representa uma remoção ou inserção, é atribuída uma pontuação negativa a esse par.

Exitem matrizes que que atribuem pontuação à comparação de cada par de bases, chamadas de matrizes de substituição. Para comparação de proteínas, as matrizes de substituição mais conhecidas são a PAM [53] e a BLOSUM [32].

Em [7], Altschul fez um estudo extensivo sobre formas de pontuar espaços em branco dentro em um Alinhamento Múltiplo de Sequências. Entre os métodos propostos estão *quasi-natural gap penalties* e *natural gap penalties*.

3.2.1 Alinhamento de Duas Sequências

Existem dois tipos básicos de alinhamentos: global e local[50]. No alinhamento global, todos os caracteres de ambas as sequências fazem parte do alinhamento. Já no alinhamento local, somente partes das duas sequências fazem parte do alinhamento. As definições de alinhamento global e local (Definições 1 e 2, respectivamente), são apresentadas em seguida.

Definição 1 *O alinhamento global de $k = 2$ palavras $S = S_1, ..., S_k$ é a associação ordenada de cada elemento de uma sequência com um elemento na outra sequência, podendo o elemento ser um elemento em branco. Ao final do alinhamento, as sequências possuem comprimento igual, l, devido à inserção de elementos em branco. Como resultado, é obtida uma tabela de k linhas e l colunas, onde o elemento na posição i da palavra S_v está na linha v e coluna i [30].*

Definição 2 *O alinhamento local de $k = 2$ palavras $S = S_1, ..., S_k$ é obtido a partir da seleção de uma subpalavra S'_i de cada $S_i \in S$. A partir daí, é realizado o alinhamento global entre essas subpalavras [30].*

Para o alinhamento de duas sequências, os métodos exatos mais utilizados são Needleman-Wunsch [63], método de alinhamento global, e Smith-Waterman [60], método de alinhamento local.

3.2.1.1 Alinhamento Global: Algoritmo Needleman-Wunsch

O algoritmo Needleman-Wunsch [63] determina o alinhamento global ótimo a partir do cálculo do alinhamento ótimo de subsequências menores, utilizando programação dinâmica [9]. Para isso é construída uma matriz F, indexada por i e j, onde $F(i,j)$ é o valor do melhor alinhamento entre os prefixos $x_{1..i}$ e $y_{1..j}$. Sabendo-se que as posições $F(i-1,j)$, $F(i,j-1)$ e $F(i-1,j-1)$ são os valores de melhor alinhamento das maiores subsequências antes de $F(i,j)$, então essas subsequências serão utilizadas para calcular o valor de $F(i,j)$ de modo a obter o maior valor possível. O valor $F(0,0) = 0$ e todas as outras coordenadas serão calculadas por meio da Fórmula 3.1:

$$F(i,j) = max \begin{cases} F(i-1,j-1) + s(x_i, y_i), \\ F(i-1,j) - d, \\ F(i,j-1) - d. \end{cases} \tag{3.1}$$

Na Fórmula 3.1, $s(x_i, y_j)$ é o valor da comparação do elemento x_i com y_j e d é o valor da comparação de qualquer elemento com um espaço em branco. A função que calcula o valor da comparação de um elemento com um espaço em branco pode ser $O + l \times E$, onde O é o valor de abertura do espaço em branco, E é o valor de extensão do espaço em branco e l é o número de espaços em brancos seguidos. Quando $O = 0$, o cálculo tem o custo do espaço em branco linear. Senão, o custo do espaço em branco é afim (*natural affine gap cost*) [64].

Para encontrar o melhor alinhamento a partir da matriz é utilizado o algoritmo *traceback*. Nesse algoritmo, a matriz é percorrida de trás, $F(n,m)$, para frente, $F(0,0)$, achando qual das coordenadas anteriores à $F(i,j)$ foi utilizada para obter o valor da coordenada atual. Se o valor utilizado foi o de $F(i-1,j-1)$, x_i e y_j são alinhados, caso o valor seja $F(i,j-1)$, x_i é alinhada com um espaço em branco representado pelo sinal '-', e caso o valor seja $F(i-1,j)$, y_j é alinhado com um espaço em branco.

A Figura 3.1 apresenta os alinhamentos globais ótimos resultantes da aplicação do algoritmo Needleman-Wunsch nas sequências GGAAT e GGAT, ilustrado na Tabela 3.1.

```
G  G  A  A  T
G  G  A  -  T

G  G  A  A  T
G  G  -  A  T
```

Figura 3.1: *Alinhamentos globais resultantes da aplicação do algoritmo de Needleman-Wunsch nas sequências GGAAT e GGAT.*

	-	**G**	**G**	**A**	**A**	**T**
-	0	$\leftarrow\ -2$	$\leftarrow\ -4$	$\leftarrow\ -6$	$\leftarrow\ -8$	$\leftarrow\ -10$
G	$\uparrow\ -2$	$\nwarrow\ 2$	$\nwarrow\ \leftarrow\ 0$	$\leftarrow\ -2$	$\leftarrow\ -4$	$\leftarrow\ -6$
G	$\uparrow\ -4$	$\nwarrow\ \uparrow\ 0$	$\nwarrow\ 4$	$\leftarrow\ 2$	$\leftarrow\ 0$	$\leftarrow\ -2$
A	$\uparrow\ -6$	$\uparrow\ -2$	$\uparrow\ 2$	$\nwarrow\ 6$	$\nwarrow\ \leftarrow\ 4$	$\leftarrow\ 2$
T	$\uparrow\ -8$	$\uparrow\ -4$	$\uparrow\ 0$	$\uparrow\ 4$	$\nwarrow\ 5$	$\nwarrow\ 6$

Tabela 3.1: *Matriz F resultante do cálculo do algoritmo de Needleman-Wunsch no alinhamento das sequências GGAAT e GGAT, com* $d = 2$, $s(x_i, y_j) = 2$ *para* $x_i = y_j$ *e* $s(x_i, y_j) = -1$ *para* $x_i \neq y_j$.

3.2.1.2 Alinhamento Local: Algoritmo Smith-Waterman

O algoritmo proposto por Smith-Waterman [60] é muito parecido com o algoritmo utilizado por Needleman-Wunsch. Na equação de recorrência, a única diferença é o acréscimo do valor zero na obtenção do máximo, que faz com que somente valores não negativos sejam obtidos (Fórmula 3.2).

$$F(i,j) = max \begin{cases} 0, \\ F(i-1, j-1) + s(x_i, y_i), \\ F(i-1, j) - d, \\ F(i, j-1) - d. \end{cases} \tag{3.2}$$

Nesse algoritmo, o alinhamento é obtido aplicando o algoritmo *traceback*, descrito anteriormente, a partir da célula com maior pontuação (célula com pontuação 5 na Tabela 3.2), até que o valor zero seja atingido.

Para o exemplo da Tabela 3.2, o melhor alinhamento local obtido pela aplicação do algoritmo Smith-Waterman é ilustrado na Figura 3.2.

G A A T
G G A T

Figura 3.2: *Alinhamento local resultante da aplicação do algoritmo de Smith-Waterman nas sequências GAATT e GGATCGA.*

	-	**G**	**A**	**A**	**T**	**T**
-	0	0	0	0	0	0
G	0	2	0	0	0	0
G	0	2	↖ 1	0	0	0
A	0	0	↖ 4	↖ 3	← 1	0
T	0	0	↑ 2	↖ 3	↖ **5**	↖ ← 3
C	0	0	0	↖ ↑ 1	↑ 3	↖ 4
G	0	2	0	0	↑ 1	↖ ↑ 2
A	0	0	↖ 4	← 2	0	0

Tabela 3.2: *Matriz resultante do cálculo do algoritmo de Smith-Waterman para o alinhamento das sequências GAATT e GGATCGA, com* $d = 2$, $s(x_i, y_j) = 2$ *para* $x_i = y_j$ *e* $s(x_i, y_j) = -1$ *para* $x_i \neq y_j$.

3.3 Alinhamento Múltiplo de Sequências

Em um alinhamento mútliplo de sequências biológicas (*Multiple Sequence Alignment - MSA*), caracteres similares pertencentes a k sequências ($k > 2$) são alinhados de maneira a ressaltar as semelhanças. Frequentemente, as sequências comparadas com MSA são biologicamente relacionadas e o objetivo é obter subpadrões conservados. À primeira vista, o problema do alinhamento múltiplo pode ser visto como uma generalização do problema do alinhamento de duas sequências (seção 3.2.1). No entanto, os biólogos usam as duas classes de alinhamentos para solucionar problemas bem distintos. Geralmente, o MSA é usado para solucionar problemas importantes e complexos como representação de famílias/superfamílias de proteínas, previsão de estrutura secundária e inferência filogenética [30].

Por ser um problema NP-completo [40], o alinhamento múltiplo tem sido abordado de diversas maneiras, de modo a encontrar soluções que levem a resultados bons com uso de memória e processamento restritos em um tempo aceitável. Essas abordagens podem ser classificadas nas categorias listadas a seguir [15, 50]:

- **Alinhamento Progressivo Global:** os métodos progressivos são executados em 3 fases. Primeiramente, são obtidos alinhamentos de todos os pares de sequências com o algoritmo Needleman-Wunsh (seção 3.2.1). Na fase 2, é construída uma árvore filogenética com a informação obtida na fase 1. Finalmente, na fase 3, a árvore filogenética construída na fase 2 é usada para guiar o alinhamento das sequências de maneira sequencial, partindo das mais fortemente relacionadas (alinhamentos dois a dois com

maior escore) até as mais fracamente relacionadas. O CLUSTAL-W [26] e o T-COFFEE [11] são exemplos de estratégias de alinhamento múltiplo que usam métodos progressivos.

- **Métodos Iterativos:** o algoritmo produz um alinhamento e o refina repetidas vezes até que o alinhamento não possa mais ser melhorado. Os métodos iterativos podem ser determinísticos ou estocáticos (não determinísticos). Os métodos determinísticos são mais simples e consistem em extrair uma sequência de cada vez de um alinhamento múltiplo e realinhá-la com as sequências restantes [29], esse processo é repetido até que o alinhamento não possa ser melhorado. Dentre os métodos estocásticos estão os Algoritmos Genéticos (AGs).

- **Outros métodos:** nessa classe, estão as categorias que possuem outras características que não estão contempladas nas categorias descritas anteriormente, podendo os métodos estarem em mais de uma categoria. Dentre as categorias estão, Métodos Estatísticos [15, 50], Métodos Baseados em Consistência [41], Métodos Baseados em Modelos [52], entre outros.

3.4 Determinação da Qualidade do Alinhamento

Para se determinar a qualidade de um alinhamento múltiplo, geralmente é utilizado o método SP (*Sum-of-Pairs*) ou alguma de suas variantes. Nesse método, todas as n sequências são combinadas em pares de modo que sejam obtidos $n * (n-1)/2$ pares. A pontução de cada par de sequências é obtida a partir da soma dos resultados da comparação dos elementos de cada coluna. A pontuação final é a soma da pontução de cada par de sequências.

A Figura 3.3 apresenta um exemplo do cálculo SP aplicado a um MSA. Na Figura, o resultado final do cáculo SP será $0-4-4=-8$, no critério de pontução onde elementos iguais recebem uma pontuação positiva de $+2$, elementos distintos recebem uma pontuação negativa de -1 e o elemento comparado com um espaço em branco recebe uma pontuação de -2, espaço em banco com espaço em branco recebe 0 de pontuação.

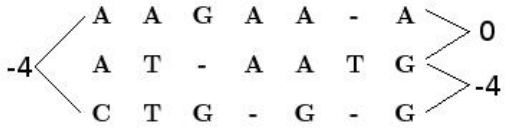

Figura 3.3: *Exemplo do cálculo SP [50].*

Altschul et. al. [45] incrementam o cálculo SP atribuindo um peso a cada par de sequências. O valor de peso multiplica a pontuação obtida no alinhamento dois-a-dois e depois disso o valor dos alinhamentos são somados. O peso

é um número obtido a partir de uma árvore filogenética montada com todas as sequências e tem o objetivo de não deixar o alinhamento tendencioso quando existem mais sequências de uma família do que de outra. Esse método é chamado de WSP (*Weighted Sum-of-Pairs*). Também foram propostos outros métodos para atribuir peso aos pares sequências no WSP, dentre eles está o método utilizado pelo CLUSTAL W [26].

3.5 Algoritmos para Alinhamento Múltiplo baseados em Algoritmos Genéticos

Nessa seção, serão detalhandos os algoritmos MSA-EA e SAGA que são os Algoritmos Genéticos mais utilizados para o problema do Alinhamento Múltiplo de Sequências.

3.5.1 MSA-EA

MSA-EA (*Multiple sequence alignment - Evolutionary Algorithm*) [43] é um algoritmo evolucionário [47] para Alinhamento Múltiplo de Sequências que se propõe a melhorar os resultados obtidos pelo algoritmo CLUSTAL V [35].

O algoritmo gera uma população inicial de alinhamentos múltiplos e, em seguida, é aplicado o operador que move as colunas de espaços em branco que existirem no alinhamento para depois do último caractere, de modo que a coluna não interfira no cálculo de aptidão. Em seguida, é feito o cálculo de aptidão. Caso o valor da aptidão não seja o desejado, é gerada uma nova população aplicando os operadores de *crossover*, mutação e seleção. O algoritmo é repetido até que a aptidão da população atinja o valor desejado. O pseudo-código do algoritmo é representado na Figura 3.4.

Algoritmo MSA-EA

```
1.  inicializar a população
2.  limpar colunas de espaços em branco
3.  avaliar
4.  enquanto o critério de parada não for satisfeito faça
5.      aplicar mutação
6.      aplicar crossover
7.      limpar colunas de espaços em branco
8.      avaliar
9.      selecionar
10. fim enquanto
```

Figura 3.4: *Algoritmo MSA-EA [43].*

3.5.1.1 Estrutura de Dados

Para representar o alinhamento múltiplo, é utilizado um *array* de sequências, onde cada sequência é uma cadeia de caracteres que contém elementos do alfabeto {C, S, T, P, A, G, N, D, E, Q, H, R, K, M, I, L, V, F, Y, W, -, X}. Os elementos {C, S, T, P, A, G, N, D, E, Q, H, R, K, M, I, L, V, F, Y, W} são os valores que os aminoácidos que compõem a cadeia de caracteres podem assumir, '-' é o símbolo que representa o espaço em branco, indicando que houve uma inserção ou deleção, e X é um símbolo indefinido, podendo ele ser qualquer uns dos valores possíveis para o nucleotídeo. A matriz resultante tem n (número de sequências) linhas e $1,5$ vezes o comprimento da maior sequência colunas, permitindo que no decorrer do algoritmo as sequências atinjam o tamanho de $1,5$ vezes o comprimento da maior sequência. No caso de ocorrência de colunas de espaços em branco, as mesmas são movidas para direita depois do último nucleotídeo, sendo desconsideradas no cálculo da aptidão.

3.5.1.2 População Inicial

A população inicial do algoritmo MSA-EA [43] pode ser gerada de três maneiras: a) aleatória, b) com o CLUSTAL V ou c) parte aleatória e parte com CLUSTAL V.

No método aleatório, a população inicial é gerada pela inicialização aleatória de colunas seguindo o procedimento : Primeiramente, para cada sequência s_i de comprimento l_i é gerada uma combinação aleatória das possíveis posições dos caracteres nas colunas da matriz (Figura 3.5.a). Os l_i primeiros elementos da combinação de colunas são selecionados e colocados em ordem crescente (Figura 3.5.b). Os elementos da sequência s_i são colocados nas posições determinadas pela combinação de colunas ordenadas na matriz e os espaços restantes são preenchidos pelo caractere '-', indicando espaço em branco (Figura 3.5.c).

O algoritmo CLUSTAL V pode ser utilizado para geração da população inicial, de forma a reduzir o número de etapas para se chegar a uma solução ótima. No entanto, esse método pode induzir o algoritmo a regiões de ótimos locais. Então, outra forma de utilizar os resultados do algoritmo CLUSTAL V é utilizá-lo para povoar somente uma pequena parte da população inicial, dessa forma, indicar ao algoritmo que existe aquela solução e ao mesmo tempo possibilitar que ele procure soluções em outras regiões.

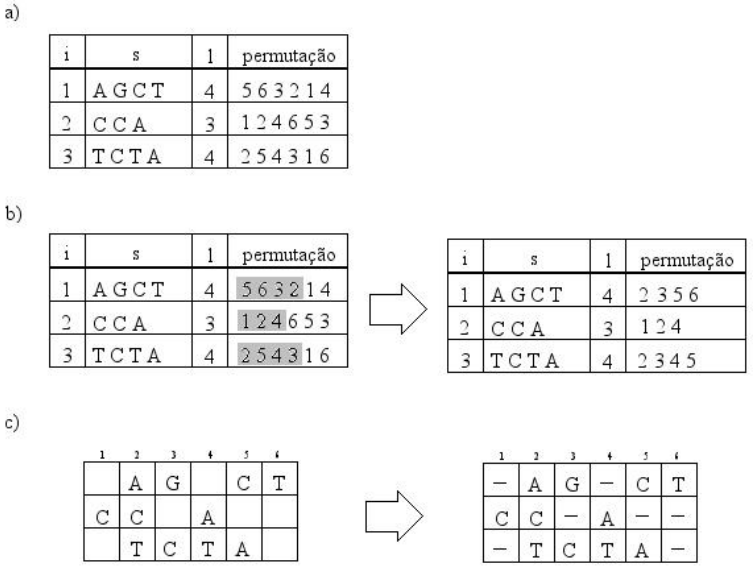

Figura 3.5: *Geração da população inicial do algoritmo MSA-EA [43].*

3.5.1.3 Avaliação

O cálculo de aptidão do MSA-EA é baseado no método SP (seção 3.4), ele utiliza a matriz de substituição PAM100 [53] e *affine gap cost* [64]. As colunas compostas somente de espaços em brancos são desconsideradas.

3.5.1.4 Geração de filhos

Durante o processo evolutivo do MSA-EA, a população é submetida a alguns operadores. A seguir são descritos os principais de forma breve [43]:

LocalShuffle: O operador seleciona um caractere aleatório de uma coluna e verifica se um dos seus vizinhos é um espaço em branco. Caso seja, é feita uma permuta de posições entre o espaço em branco e o caractere. Caso ambos os vizinhos sejam espaços em branco, é escolhido de forma aleatória o vizinho que será alvo de permuta.

BlockShuffle: O operador é semelhante ao *LocalShuffle*. O operador seleciona um conjunto de caracteres posicionados em colunas consecutivas (bloco) e o movimenta para direita ou para esquerda dependendo de onde houver espaço em branco. Caso ambos os vizinhos sejam espaços em branco é escolhido de forma aleatória a posição que o bloco deverá se mover.

GrowMatchedColumns: O operador seleciona uma coluna em que todos os caracteres são iguais. Se possível, o operador tenta trocar os elementos da coluna com espaços em brancos vizinhos de modo a formar outras colunas de elementos iguais diretamente à esquerda ou direita da coluna selecionada.

RecombineMatchedColumns: O operador seleciona dois alinhamentos da população e de forma aleatória, depois seleciona em cada alinhamento uma coluna com todos os caracteres iguais, sendo os caracteres da coluna do alinhamento 1 diferente dos da coluna do alinhamento 2. O operador tenta criar um alinhamento pela troca de espaços em branco por caracteres, de modo que o filho contenha ambas as colunas.

CleanUpGapColumns: Operador move todas as colunas de espaços em branco para o lado direito após o último caractere. O operador é aplicado sempre após que um dos operadores acima é aplicado.

3.5.2 SAGA

O SAGA (*Sequence Alignment by Genetic Algorithm*) [33] é um Algoritmo Genético para Alinhamento Múltiplo de Sequências, que segue as mesmas etapas do Algoritmo Genético Típico, ilustrado na Figura 2.1 da seção 2. O pseudo-código do SAGA é representado na Figura 3.6 e seu modelo de funcionamento é ilustrado na Figura 3.7.

Algoritmo SAGA

Inicialização	1. **Criar** G_0
Avaliação	2. **Avaliar** a população da geração n (G_n)
	3. **Se** a população está estabilizada **então** END
	4. **Selecionar** indivíduos para compor G_{n+1}
	5. **Avaliar** os filhos esperados (EO)
Breeding	6. **Selecionar** os pais de G_n
	7. **Selecionar** o operador
	8. **Gerar** um novo filho
	9. Manter ou eliminar o novo filho para a geração G_{n+1}
	10. **Ir para** 6 até que todos os filhos tenham sido colocados com sucesso na geração G_{n+1}
	11. $n = n + 1$
	12. **Ir para** AVALIAÇÃO
End	13. end

Figura 3.6: *Algoritmo SAGA [33].*

O SAGA gera uma população inicial aleatoriamente (G_0), que é composta de indivíduos que são Alinhamentos Múltiplos de Sequências (Figura 3.6, linha 1). Em seguida, a população gerada é avaliada (linha 2). A metade mais apta da população é selecionada para compor a próxima geração (linha 4). A partir dos

valores de aptidão da população são gerados valores que indicam o número de filhos esperados (EO) para cada indivíduo da população (linha 5). Os pais são selecionados pelo método Roda Roleta (seção 2.3), de acordo com o valor de EO de cada indivíduo (linha 6). Também pelo método da Roda Roleta é selecionado o operador de *crossover* ou mutação que será aplicado aos pais (linha 7). O operador selecionado é aplicado ao(s) pai(s) que gerarão novo(s) filho(s) (linha 8). Os filhos gerados são inseridos na população se forem aptos o suficiente e se não forem iguais à nenhum indivíduo já presente na população (linha 9). O processo de geração de filhos é repetido até que a nova população esteja completa (linha 10). Em seguida o processo completo é repetido gerando novas populações até que o critério de parada seja satisfeito.

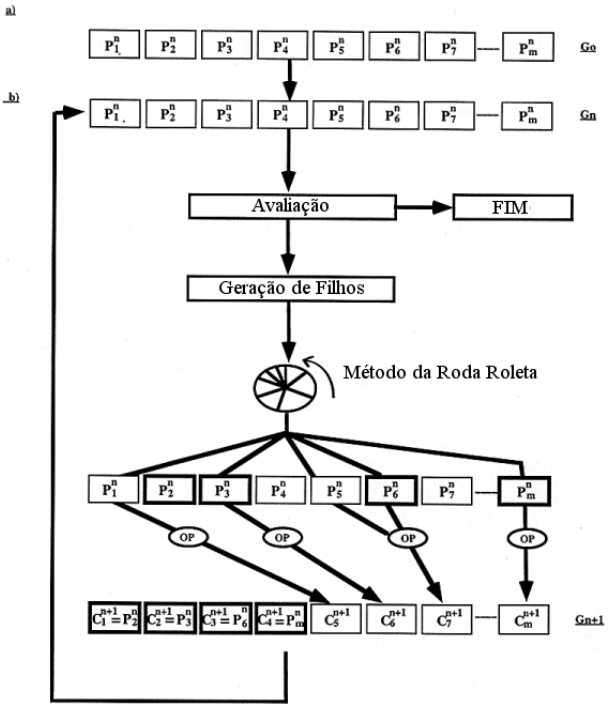

Figura 3.7: *O layout do algoritmo SAGA [33].*

Na Figura 3.7, as caixas P_1^n a P_m^n representam os pais na geração n, sendo C_1^{n+1} a C_m^{n+1} seus filhos. As caixas em negrito C_1^{n+1} a C_4^{n+1} indicam metade mais apta da população é selecionada para compor a próxima geração.

3.5.2.1 População Inicial

A primeira etapa do algoritmo consiste em gerar uma população inicial de forma aleatória. Essa população é formada por conjuntos de alinhamentos com somente espaços em branco terminais (não existe espaços em branco dentro da sequência). A estrutura de dados utilizada para representar o alinhamento é um *array* de duas dimensões, onde cada linha representa uma sequência alinhada

e cada célula contêm um caractere da sequência ou um espaço em branco. As estruturas de dados possuem um tamanho fixo, não variando entre si.

Ao gerar um alinhamento, é atribuído um número aleatório (*offset*) a cada sequência do alinhamento, sendo cada sequência movida para direita o número de espaços correspondente ao número aleatório gerado. Para igualar o comprimento das sequências do alinhamento, são inseridos espaços em branco ao final das mesmas até o o comprimento do *array* seja completo. É importante ressaltar que a população inicial e das populações que a seguem não possuem elementos duplicados e que o *offset* geralmente varia de 0% a 25% do comprimento da maior sequência.

3.5.2.2 Avaliação

No SAGA, foram utilizadas duas funções de avaliação: 1) uma funcão de avalição WSP (seção 3.4) com a função de peso do CLUSTAL-W [26], utilizando *quasi-natural gap penalties* [7] e a matriz de substituição PAM250 [53]; e 2) uma funcão de avalição WSP (seção 3.4) com a função de peso proposta por Altschul et. al. [45], utilizando *natural gap penalties* [7] e a matriz de substituição PAM250 [53]. Uma das funções é selecionada para ser utilizada em determinada execução do algoritmo.

Para diminuir os erros, os valores dos alinhamentos obtidos na função de avaliação são transformados em valores normalizados conhecidos como EO (filhos esperados), que indicam quantos filhos é provável o alinhamento ter. A metade mais fraca da população é substituída por novos filhos e a mais forte é mantida formando a nova geração, sendo esse processo é chamado de gerações sobreescritas (*overlapping generations*) [19].

3.5.2.3 Seleção

Primeiramente, 50% da população mais apta de pais é selecionada para compor a próxima geração. Em seguida, são gerados os filhos para compor a outra metade da nova geração. Os filhos são gerados a partir dos pais, sendo que os pais são selecionados de acordo com seu valor de EO por meio do método Roda Roleta (seção 2.3). Cada vez que um pai é selecionado seu valor de EO é reduzido. Depois que todos os pais foram selecionados, o operador que será aplicado a eles para geração de filhos é também selecionado por meio do método de Roda Roleta. Os operadores são colocados na roleta de acordo com sua probabilidade de ocorrência. O operador selecionado é aplicado ao(s) pai(s) gerando novo(s) filho(s). Os filhos gerados são avaliados, se não forem iguais à nenhum elemento que já está na população e possuírem um valor de aptidão alto o suficiente, eles serão inseridos na nova população. Senão, ele é eliminado e um novo filho é gerado. Essas verificações procuram garantir respectivamente a qualidade e a diversidade da nova população.

Esse processo é repetido até que o critério de parada seja atingido. O critério de parada normalmente utilizado é a estabilização da qualidade da população por um determinado número de gerações, tipicamente 100.

3.5.2.4 Geração de filhos

O SAGA [33] implementa 22 operadores, divididos em *crossovers* e mutações. O algoritmo não diferencia um operador de outro, sendo cada um escolhido de forma aleatória por meio do método da Roda Roleta (seção 2.3). Os parâmetros a serem utilizados nos operadores também são escolhidos de forma aleatória.

A seguir serão descritos os principais operadores implementados pelo SAGA.

Crossover

Operadores de *crossover* mais utilizados no SAGA são:

- **Crossover de** 1 **ponto**: No *crossover* de 1 ponto, é selecionada uma posição aleatória onde o pai 1 será dividido. O pai 2 é dividido de modo que seu lado direito possa ser unido com o lado esquerdo do pai 1 e vice-versa, formando dois novos filhos. Qualquer espaço vazio que apareça nos pontos de junção dos lados é preenchido com sinal de nulo, sendo que os operadores de mutação tratarão esses sinais. O melhor filho é selecionado para integrar a população. O *crossover* de 1 ponto exemplificado na Figura 3.8.

- **Crossover uniforme**: O *crossover* uniforme foi desenhado para que dois pais pudessem trocar algumas regiões. Primeiramente, são selecionadas posições consistentes em ambos os pais, isto é, selecinar colunas compostas pelos mesmos elementos nos dois alinhamentos pais. Os blocos entre essas posições consistentes são trocados formando novos alinhamentos, os filhos. O *crossover* uniforme é exemplificado na Figura 3.9.

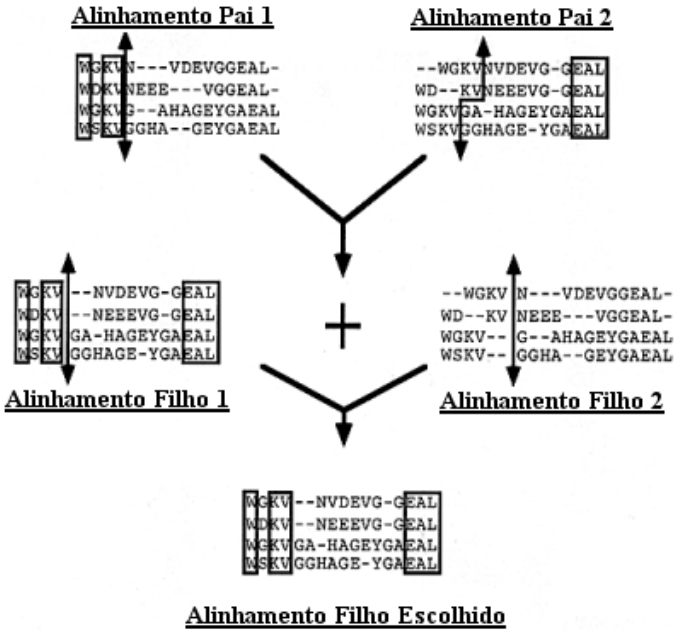

Figura 3.8: *Crossover de um* 1 *entre dois alinhamentos produzindo dois filhos [33].*

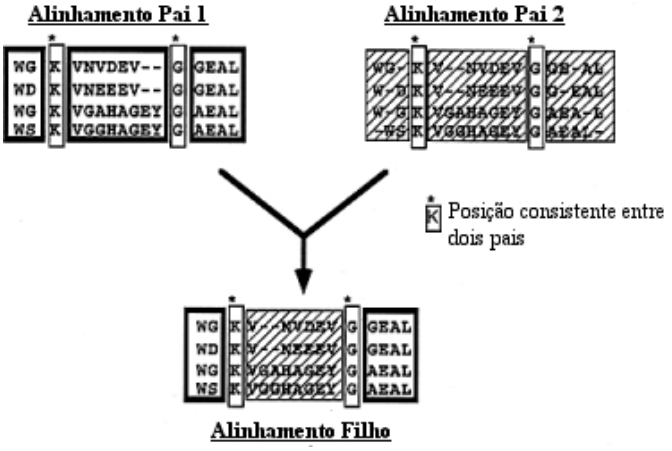

Figura 3.9: *O crossover uniforme [33].*

Mutação

Operadores de mutação mais comumente utilizados no SAGA são:

- **Inserção de espaços em branco**: O operador extende o alinhamento, inserindo espaços em branco. Para manter o alinhamento das sequências, são inseridos o mesmo número de espaços em branco em cada uma delas. As

sequências são divididas em dois grupos. Dentro de um grupo, os espaços em branco são inseridos na mesma posição em cada sequência. O processo é repetido dividindo as sequências em grupos ainda menores. Essa divisão é inspirada na árvore filogenética, como se cada nó de uma árvore fosse um grupo. O processo é ilustrado pela Figura 3.10.

- **Deriva de blocos (*Block Shuffling*)**: O operador deriva de blocos desloca blocos de espaços em branco ou de caracteres. No SAGA [33], o bloco de caracteres é definido como subsequências que se sobrepõem em uma ou mais sequências. Cada subsequência delimitada por um espaço em branco ou o término de uma sequência e, similarmente, um bloco de espaços em branco é um conjunto de espaços em branco que se sobrepõem [33]. Primeiramente, é selecionado um caractere ou espaço em branco e depois o bloco é derivado dessa escolha. Esses blocos podem ser movidos dentro do alinhamento como mostrado na Figura 3.11 (b,c,d). Os limites desse movimento são o próprio alinhamento. Um espaço em branco só pode ser movido até que ele encontre outro espaço em branco. Similarmente, a subsequência de caracteres pode ser movida até que encontre outra subsequência de caracteres. Essas operações podem ser utilizadas da seguinte forma:

  - Mover blocos completos (Figura 3.11.a).

  - Partir o bloco horizontalmente e mover um dos sub-blocos para esquerda ou para direita (Figura 3.11.c).

  - Partir o bloco verticalmente e mover uma metade para a esquerda ou para direita (Figura 3.11.d).

- **Pesquisa por blocos (*Block Searching*)**: Dada uma subsequência, a pesquisa por blocos encontra o bloco a que ela pertence. Primeiramente, é selecionada uma subsequência de comprimento aleatório, em uma posição aleatória de uma sequência. Então, subsequências do mesmo tamanho de outras sequências são comparadas com a subsequência selecionada, sendo que a mais parecida é escolhida e adicionada à primeira, de forma a formar um pequeno perfil (*profile* [22]). Nas sequências restantes é repetido o mesmo processo e a subsequência mais próxima é adicionada ao perfil. O processo é repetido iterativamente até que tenha sido selecionada uma subsequência de cada sequência. As sequências são movidas de modo a formar um bloco com as subsequências no alinhamento.

- **Rearranjo do ótimo local ou do subótimo (*Local optimal or sub-optimal rearrangement*)**: Existem situações em que a presença de ótimos locais muito estáveis dificulta aplicação dos outros operadores. O operador de rearranjo reposiciona o espaços em branco de modo a evitar que os alinhamentos fiquem presos em ótimos locais. Ele faz isso de duas formas: a) por estudo exaustivo do posicionamento dos espaços em branco dentro dos blocos ou b) por um AG alinhamento local (LAGA) [27]. O estudo exaustivo é feito se é preciso examinar um número combinações menor que um número determinado (tipicamente 2000). Se não, o LAGA é executado.

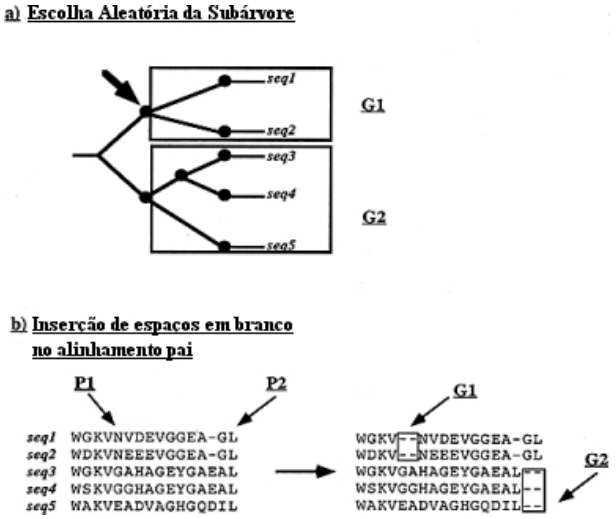

Figura 3.10: *Operador de mutação do SAGA, inserção de espaços em branco [33].*

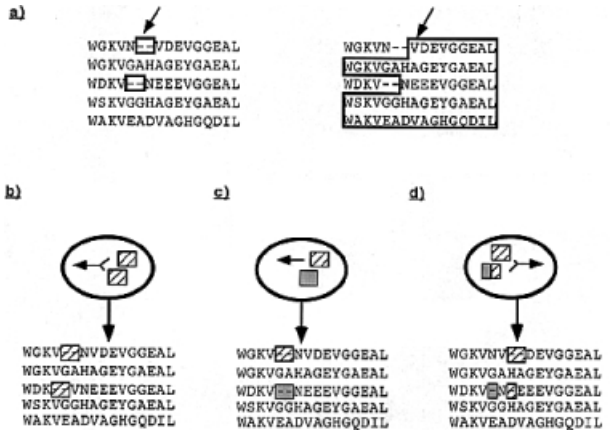

Figura 3.11: *Operador de mutação do SAGA, deriva de blocos [33].*

Capítulo 4

Algoritmos Genéticos Paralelos para Alinhamento Múltiplo de Sequências

4.1 iPGA

Nessa seção será descrita a estratégia de paralelização do Algoritmo Genético para Alinhamento Múltiplo de Sequências chamado iPGA (*island Parallel Genetic Algorithm*) [58, 57]. O iPGA é um Algoritmo Genético paralelo que utiliza o modelo da ilha (seção 2.6) como estratégia de paralelização. Em cada ilha é executado AG que, em intervalos pré-definidos, troca seus melhores alinhamentos com os melhores alinhamentos dos seus vizinhos lógicos, de modo que ao final da troca todos possuam o mesmo número de alinhamentos que possuíam anteriormente.

Os algoritmos sequenciais executados em cada ilha possuem seu funcionamento semelhante ao do algoritmo SAGA (seção 3.5.2), com duas pequenas diferenças, uma no cálculo de aptidão e outra nos operadores implementados.

O cálculo de aptidão utilizado é o WSP (seção 3.4) que utiliza o método *rationale 1* desenvolvido por Altschul et. al. [45] para calcular o peso dos pares de sequências. A matriz de substituição escolhida foi a PAM250 [53] e a função de custo de espaços em branco foi a *natural affine gap* [23].

Os operadores implementados estão dentro do conjunto de operadores implementados no SAGA (seção 3.5.2). Foram implementados dois operadores de *crossover*, o uniforme e o de 1 ponto; e três operadores de mutação, inserção de espaço em branco, deriva de blocos e pesquisa por blocos.

4.1.1 Resultados

O algoritmo descrito nessa seção foi apresentado em [58, 57]. Ambos os artigos apresentam o projeto do iPGA, porém os resultados apresentados foram para testes distintos.

Os resultados foram obtidos na máquina PARAM 10000 [42], um *cluster* de 40 nodos, onde cada nodo (*SUN Ultra450*) é composto por quatro processadores de 300 MHz e uma memória compartilhada de 512 MB. O algoritmo foi imple-

mentado na linguagem C e utiliza uma biblioteca para máquina virtual paralela (PVM). A máquina não foi disponibilizada com exclusividade para o algoritmo, tendo ele que dividir os recursos de processamento e memória com outros processos.

Em [57], um conjunto de quatro casos de teste (Dfr, Gcr, Globin e S protease) foi escolhido do Banco de Dados de Alinhamento Estrutural Pascarella [8]. Os resultados da execução do algoritmo foram comparados aos do algoritmo CLUSTAL-W [26]. Em todos os casos o algoritmo obteve uma pontuação melhor que a do CLUSTAL-W.

Também foi feita uma análise da variação da qualidade do alinhamento em relação à variação dos parâmetros de migração. Os parâmetros são intervalo de migração, número de elementos migrados e estratégia de seleção de emigrantes. A estratégia de seleção de emigrantes é sempre aleatória, enquanto o intervalo de migração e o número de elementos variam. Foram utilizados na comparação os resultados da execução de um Algoritmo Genético Sequencial (AGS) que alinhou o mesmo número de elementos e executou o mesmo número de gerações.

Os resultados obtidos pelo iPGA foram sempre melhores que os resultados obtidos pelo AGS, mostrando que a evolução separada das subpopulações aumenta a probabilidade de uma das populações evoluir em direção do melhor resultado. Os resultados são melhores quando poucos elementos são migrados. O melhor resultado foi para a migração de 1 elemento em 50 gerações. No entanto, a migração de poucos elementos em intervalos demasiado longos prejudica o desenvolvimento populacional, pois a variabilidade genética em cada nó é afetada, ficando a evolução limitada a um pequeno espaço com os mesmos indivíduos.

Apesar de se tratar de um algoritmo sequencial (CLUSTAL-W) e outro paralelo (iPGA) e ainda estarem em um contexto onde os processadores não estão dedicados à aplicação, foram coletados dados do tempo de execução dos algoritmos. O iPGA obteve resultados no mínimo 50 vezes melhores que o CLUSTAL-W.

Em [58], também foram foram escolhidos quatro casos de teste (Acprot, Globin A, Globin B e Sprot) do Banco de Dados de Alinhamento Estrutural Pascarella [8]. Os resultados do algoritmo foram comparados aos resultados do Algoritmo Genético Sequencial (AGS) [58] e do CLUSTAL-W [26]. Novamente, os resultados do iPGA foram melhores que os dos outros algoritmos. O iPGA obtém o melhor resultado em no mínimo 50% das vezes.

Na análise dos resultados em relação à variação dos parâmetros de migração, os resultados obtidos pelo iPGA foram melhores que os resultados obtidos pelo AGS, ratificando os resultados obtidos em [57]. A melhor média obtida foi para o intervalo de migração 25 e 4 elementos migrados. Os resultados obtidos em [57] e em [58] mostram que a variação dos parâmetros de migração tem direta relação com a qualidade da população final. Logo, é interessante que algumas configurações desse parâmetros sejam testadas de modo a utilizar a que produz melhores resultados.

4.2 PHGA

Essa seção descreverá o algoritmo PHGA (*Parallel Hybrid Genetic Algorithm* descrito em [64, 65]. O PHGA é um Algoritmo Genético que utiliza a estratégia de ilha para paralelização. Em cada ilha, é executado um Algoritmo Genético do tipo GENITOR [61].

O AG do tipo GENITOR cria uma população inicial aplicando repetidas vezes às sequências a serem alinhadas uma versão modificada do alinhamento progressivo (Figura4.1, linha 2). Depois, ele ordena os alinhamentos obtidos do mais apto (menor valor) ao menos apto (maior valor). Então, ele verifica se aplicará o operador de mutação ou o de *crossover*. Se for o operador de mutação (linha 6), ele seleciona o alinhamento mais apto (p) (linha 7) e aplica o operador de mutação ao alinhamento, dando origem ao filho c (linha 8). Se c for mais apto que p, ele substitui p (linha 9). Se o operador de *crossover* for o selecionado, ele é aplicado aos dois alinhamentos mais aptos (p_1 e p_2) gerando um filho c (linha 12). Se c for mais apto que o elemento menos apto da população, ele é inserido na mesma na ordem correspondente e o elemento menos apto é eliminado (linha 13). A etapa de aplicação de operadores é novamente executada até que a população tenha convergido ou até que o valor limite de execuções tenha sido atingido. Os melhores elementos da população são migrados entre os processadores durante a execução de acordo com os intervalos de migração (linha 16).

Algoritmo PHGA

1. **Para** cada população de cada ilha **faça**
2. **Inicializar** população
3. **Fim Para**
4. **Enquanto** (não convergir) ou (não atingir número máximo de gerações) **faça**
5. **Para cada** população de cada ilha **faça**
6. **Se** (valor_aleatório() < taxa_mutação) **faça**
7. **Selecionar** pai(p) mais bem classificado
8. c = **mutação**(p)
9. **Substituir** filho(c) pelo pai(p) **se** c mais apto que p
10. **Se não**
11. **Selecionar** pais(p_1, p_2) mais bem classificados
12. c = ***crossover***(p_1, p_2)
13. **Inserir** filho c na população substituindo-o pelo menos apto, caso ele seja mais apto que o menos apto
14. **Fim Se**
15. **Fim Para**
16. **Migrar** elementos no intervalo de migração determinado
17. **Fim Enquanto**

Figura 4.1: *Algoritmo PHGA [65].*

A PHGA não resolve o problema do alinhamento diretamente. Ao invés disso,

ele tenta encontrar o menor caminho em um grafo acíclico direcionado com pesos nas arestas e assim encontra o melhor alinhamento. As sequências a serem alinhadas são representadas por um grafo k-dimensional, onde k é o número de sequências. A Figura 4.2 mostra o exemplo de um grafo de duas sequências, onde o melhor alinhamento consiste no caminho de menor custo entre S e T. Em um grafo de duas sequências, em uma posição aleatória P (Figura 4.2.b), são possíveis três caminhos diferentes, o caminho horizontal (número 1), o vertical (número 2) e o diagonal (número 3).

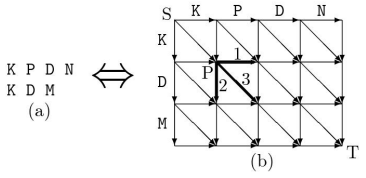

Figura 4.2: *(a) Alinhamento dois-a-dois do PHGA. (b) Grafo resultante. [64]*

O alinhamento, na Figura 4.3.a, pode ser representado pelo grafo em b ou por c. O comprimento do alinhamento pode variar dependendo do caminho escolhido.

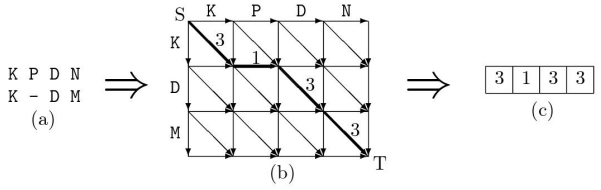

Figura 4.3: *Alinhamento final obtido pelo PHGA [64].*

A qualidade do alinhamento é calculada pela função de aptidão SP (seção 3.4) utilizando a matriz de substituição PAM250 [53] e a função de custo de espaço em branco *quasi-nature* [23].

O operadores de *crossover* e mutação foram adaptados para a estrutura de dados utilizada.

O operador *crossover* produz um novo alinhamento a partir da junção de outros dois alinhamentos, conforme ilustrado no exemplo da Figura 4.4. Primeiramente, dois pais, *Pai-1* e *Pai-2* (Figura 4.4.a e b), são escolhidos. As suas representações de alinhamento são "3 1 3 3"e "2 3 1 3 1". Então, um ponto P de *crossover* é escolhido aleatoriamente no caminho do alinhamento do *Pai-1* (Figura 4.4.a). Depois, o alinhamento de S até P do *Pai-1* é copiado para o Filho temporário e, então, o alinhamento do *Pai-2* é copiado de trás para frente até que um alinhamento possa ser garantido. O Filho temporário gerado é "3 1 $*$ 3 3"(Figura 4.4.c). Finalmente, o segmento que falta é gerado aleatoriamente, criando um novo alinhamento "3 1 2 3 3"(Figura 4.4.d).

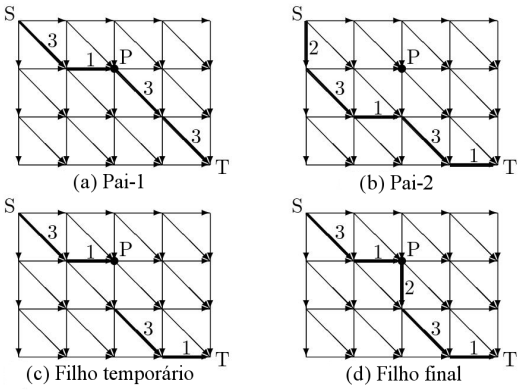

Figura 4.4: *PHGA crossover [64].*

O operador mutação produz uma pequena alteração no alinhamento. Um exemplo é ilustrado na Figura 4.5, onde o alinhamento "3 1 3 3"(Figura 4.5.a) é o pai e o alinhamento "3 3 1 3"(Figura 4.5.b) é o filho resultante. Primeiro, o operador seleciona um pequeno segmento do pai de comprimento aleatório (segmento "1 3", entre P e Q). Depois, ele aplica uma pequena mutação ao segmento de forma que esse novo segmento gerado (segmento "3 1") compartilhe os mesmos pontos de início (P) e de fim (Q).Então, ele substitui o novo segmento pelo antigo.

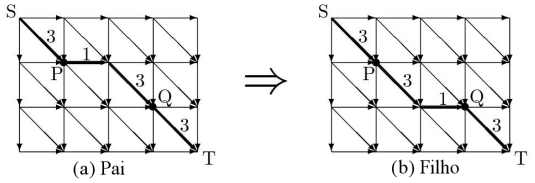

Figura 4.5: *PHGA mutação [64].*

4.2.1 Resultados

O desempenho do PHGA foi comparado com outros algoritmos de Alinhamento Múltiplo de Sequências: MSA [4], OMA [4], versão 0.95 do SAGA [33] e o ITER [64]. O ambiente utilizado como referência para os testes foi a máquina *Sun Ultra* 80 com quatro processadores de 450-MHz e 1 GB de memória.

Nas comparações, a qualidade do alinhamento foi determinada pelo cálculo que mede a porcentagem que o valor do alinhamento excede o valor do alinhamento ótimo (quanto menor o valor, melhor a qualidade do alinhamento). Como não se sabe o valor ótimo do alinhamento, é definido um limite inferior (S^*_{ij}) para cada alinhamento dois-a-dois (s_i e s_j). Logo $L = \sum_{i<j} S^*_{ij}$ é o limite inferior para o escore do Alinhamento Múltiplo de Sequências e $PEL = 100 \times (real\_cost - L)/L(\%)$ é a qualidade do alinhamento.

Os algoritmos foram testados nas 5 referências (*reference 1-5*) da BAliBASE (*Benchmark Alignment dataBASE*) [54]. Os algoritmos MSA e OMA foram submetidos somente aos *benchmarks* do *reference 1*, pois eles são leves o suficiente para esses algoritmos processarem. Como o MSA não conseguiu obter resultados nos *benchmarks* 1uky e ga14, somente os 80 *benchmarks* restantes foram usados na comparação.

Para todos os *benchmarks*, o tamanho total da população foi estabelecido como 400 indivíduos, divididos igualmente entre 8 ilhas. O critério de parada do algoritmo era 30000 gerações ou 3000 gerações sucessivas em que não houvesse alteração do melhor resultado. O intervalo de migração utilizado foi 500 gerações e o tamanho da população migrada foi 5 dos melhores indivíduos.

Os resultados obtidos pelo PHGA, utilizando o cálculo PEL, foram melhores que os dos outros algoritmos no *reference 1*.

Considerando o tempo de CPU, o PHGA é aproximadamente 10 vezes mais rápido que o SAGA, 15 vezes mais rápido que o OMA, 200 vezes mais rápido que o MSA e, apesar de ser uma abordagem paralela, ele é por volta de $1,5$ vezes mais lento que o ITER. Em sua abordagem sequencial, o PHGA é em média 3 vezes mais lento que sua abordagem paralela (resultado para 4 processadores), sendo ainda mais rápido que o SAGA, OMA e o MSA. Outra vantagem do PHGA é que ele ocupa menos memória. Enquanto o MSA e o OMA utilizaram 1 GB e $1,5$ GB para alguns *benchmarks* difíceis, o PHGA não ocupou mais de 100 MB em nenhum dos testes realizados.

O resultado PEL dos algoritmos PHGA, ITER e SAGA nas 5 referências (*reference 1-5*), mostrou que o PGHA continuou resultados de melhor qualidade e em termos de tempo de processamento. Ele continuou mais rápido que o SAGA (aproximadamente 7 vezes mais rápido) e mais lento que o ITER (aproximadamente $0,84$ vezes mais lento).

Para verificar a escalabilidade do PHGA foram feitos testes em três *benchmarks* mais pesados do *Reference 2*. O *benchmark* da lcpt tem 15 sequências de comprimento máximo 434 e gastou o tempo $1.399,85$ segundos. O *benchmark* da lasjA tem 20 sequências de comprimento máximo 389 e gastou o tempo $2.781,07$ segundos. O *benchmark* da 1lv1 tem 24 sequências de comprimento máximo 473 e gastou o tempo $6.100,92$ segundos. O algoritmo conseguiu escalar de forma que o tempo gasto não foi alto demais. No entanto, não se pode concluir que o ele será escalável para problemas maiores.

Para avaliar a qualidade do alinhamento do ponto de vista biológico, foi utilizado o programa *bali_score*, fornecido pelo BAliBASE [54, 1]. Foi verificado que o PHGA obteve resultados piores que o CLUSTAL-W, PRRP, T-COFFEE e métodos FFT-NS-i [48]. Considerando a possibilidade dos resultados terem sido piores pelo fato do PHGA está usando uma função de aptidão simples, o PHGA foi executado novamente introduzindo na função de avaliação pesos estabelecidos pelo método *rationale 2* desenvolvido por Altschul et. al. [45] e foi removida a penalidade dos espaços em branco terminais (espaços em branco após o último caractere da sequência). Os novos resultados foram comparáveis aos dos algoritmos CLUSTAL-W e PRRP e um pouco piores que os métodos T-COFFEE e FFT-NS-i. Os autores acreditam que os resultados ainda podem ser melhorados com a implementação de uma função de avaliação mais adequada.

4.3 QGA

Essa seção apresenta o QGA (*Quantum Genetic Algorithm*), um Algoritmo Genético quântico para o problema do MSA [49]. O QGA segue a estrutura do AG típico, ilustrado na seção 2 na Figura 2.1.

O QGA (Figura 4.6), primeiramente, cria uma população quatro cromossomos (QPOP - quatro matrizes quânticas) a partir das sequências que serão alinhadas (linha 1). Depois, ele computa um alinhamento inicial usando o método progressivo de Feng e Doolittle [20] (linha 2). Esse alinhamento é tido como a melhor solução até que seja encontrada uma solução melhor que ela (linha 3). O alinhamento inicial é convertido em uma Matriz Binária (MB). Então, na etapa de geração de filhos, são aplicados os operadores de interferência e *crossover* na QPOP gerando uma população de 16 cromossomos (linhas 5 e 6). O *crossover* é seguido pela mutação quântica, onde um dos operadores de mutação é escolhido aleatoriamente é aplicado (linha 7). A próxima operação é a de medida, gerando uma MB a partir de cada cromossomo (linha 8). As MBs são então traduzidas em matrizes alfabéticas (linha 9) e avaliadas pela função COFFEE [34] (linha 10). Se alguma das soluções for melhor que a solução inicial, ela a substitui (linha 11). Para compor a população da próxima geração, são selecionados os três cromossomos que obtiveram melhores resultados na avaliação e o quarto cromossomo é selecionado aleatoriamente. A etapa de geração de filhos é repetida até que o critério de parada seja atingido (linha 12).

Algoritmo Genético Quântico (QGA)

1. **Gera** uma população com 4 cromossomos
2. **Gera** um alinhamento inicial Aln_0. Sendo BM a matriz binária correspondente
3. **Estabelece** $Aln_{best} = Aln_0$ e $C_{best} = C(Aln_0)$

4. **Repete**
5. **Aplicar** interferência de acordo com a melhor solução
6. **Aplicar** *crossover*
7. **Aplicar** mutação
8. **Aplicar** operador de medida em cada cromossomo gerando um BM de cada um (BM_i)
9. **Traduzir** cada (BM_i) em uma Aln_i
10. **Avaliar** cada Aln_i
11. **Se** $C(Aln_{best}) < C(Aln_i)$ **então** $Aln_{best} = Aln_i$ e $C_{best} = C(Aln_i)$
12. **Até** critério de parada seja satisfeito

Figura 4.6: *Algoritmo Genético Quântico (QGA) [49].*

De acordo com a computação quântica, um bit quântico (*qubit*) pode estar em uma uma sobreposição de estados. Então um *qubit* (*quantum bit*) pode ser definido como uma partícula que pode assumir os estados 0, 1 ou a sobreposição dos dois estados ao mesmo tempo.

O Alinhamento Múltiplo de Sequências pode ser visualizado como uma matriz binária - MB (Figura 4.7), onde:

- Cada linha representa uma sequência no alinhamento;
- O número 1 representa um caractere na mesma coordenada na sequência e 0 a ocorrência de um espaço em branco;

Figura 4.7: *Representação binária do Alinhamento Múltiplo de Sequências [49].*

Na computação quântica, cada sequência é representada por meio de um registrador quântico (Figura 4.8). Cada elemento da sequência é representado por um *qubit* $\begin{pmatrix} a_i \\ b_i \end{pmatrix}$, sendo os valores de a_i e b_i são delimitados pela fórmula $\mid a_i \mid^2 + \mid b_i \mid^2 = 1$. Sendo $\mid a_i \mid^2$ e $\mid b_i \mid^2$ as probabilidades da ocorrência de um caractere e um espaço em branco respectivamente. Como resultado final é obtida uma matriz de probabilidade (Figura 4.9). Essa matriz possibilita o cálculo da probabilidade de ocorrência de cada configuração possível da MB (Figura 4.7), isto é, de cada alinhamento possível.

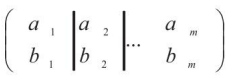

Figura 4.8: *Representação quântica de uma sequência [49].*

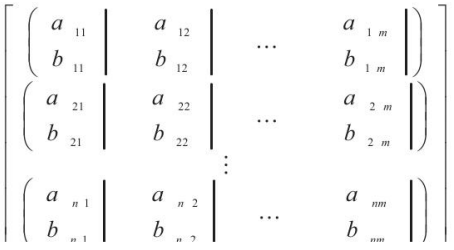

Figura 4.9: *Matriz quântica apresentando o Alinhamento Múltiplo de Sequências [49].*

Os operadores aplicados utilizados no algoritmos são explicados com maior detalhe, a seguir:

A operação de medida (*The operation of measure*)

O operador de medida transforma por projeção a matriz quântica em uma MB. Esse operador permite a retirada de uma possível solução dentre as que existem na matriz quântica, sem modificá-la (Figura 4.10). Depois, a MB é traduzida em uma matriz alfabética (Figura 4.11), isto é, substitui os zeros (0) por espaços em branco e uns (1) por caracteres.

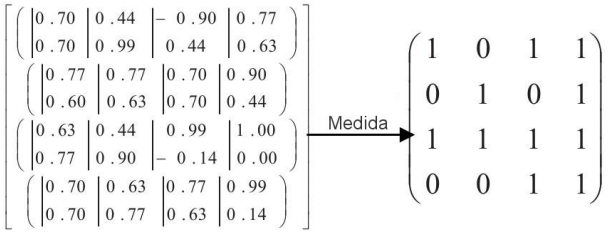

Figura 4.10: *Operação de medida na matriz quântica [49].*

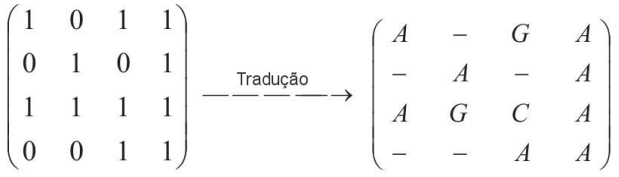

Figura 4.11: *Tadução de uma matriz binária (MB) em matriz alfabética [49].*

Interferência Quântica

O operador de interferência quântica aumenta a amplitude das soluções boas e diminui a amplitude das soluções ruins. Ele faz isso alterando o valor de cada *qubit* de forma a aumentar probabilidade de obtenção (pela operação de medida) de uma solução mais próxima das soluções boas.

Mutação

O operador de mutação consiste em basicamente alterar os *qubit*s dentro do cromossomo. Os operadores de mutação são uma adaptação dos operadores descritos no algoritmo SAGA [33], descrito na seção 3.5.2.

Crossover

Os operadores de *crossover* (*crossover* de 1 ponto e *crossover* uniforme) utilizados no Algoritmo Genético são inspirados nos operadores do algoritmo SAGA [33], descrito na seção 3.5.2.

4.3.1 Resultados

O QGA foi implementado em MATLAB 7 e testado em computador pessoal. Foram utilizadas as bases do *Reference 1-5* da BAliBASE [54] versão dois para realização dos testes. Os algoritmos CLUSTAL, DIALIGN e T-COFFEE também foram executados no ambiente de teste para as mesmas bases e os resultados obtidos por eles foram comparados aos do QGA. São utilizados os cálculos SP e pontuação por coluna - PC (proporção de colunas alinhadas corretamente) proposto na BAliBASE para medir a qualidade dos alinhamentos.

Os resultados obtidos pelo QGA tiveram uma pontuação média próxima ou superior aos resultados apresentados pelos outros algoritmos. Na *Reference 1* para o cálculo SP, o QGA obteve resultados superiores aos outros algoritmos, enquanto nas outras referências o resultado de SP é próximo aos dos outros algoritmos. Nas *Reference 4-5*, os resultados do cálculo PC do QGA foram ligeiramente inferiores aos do DIALIGN e aos do T-COFFEE. Comparando os resultados do QGA com os de outros AGs, os resultados obtidos pelo QGA foram melhores na maioria dos casos. A qualidade se deve à implementação e a aplicação dos operadores e especialmente ao uso da função de aptidão COFFEE.

Em comparação com AGs clássicos, o QGA reduz significativamente o tamanho da população e o número de iterações para se obter a solução boa.

4.4 DC-GA

Nessa seção será descrita uma estratégia para solucionar o problema do Alinhamento Múltiplo de Sequências de DNA. A estratégia DC-GA proposta [44] é um algoritmo que utiliza Algoritmos Genéticos [27] conjuntamente com técnicas de divisão e conquista [36] para solucionar o problema.

O algoritmo é divido em três fases. Primeiramente, o algoritmo recebe como entrada um conjunto de sequências para serem alinhadas. Depois, na primeira fase, ele aplica ao conjunto de sequências um Algoritmo Genético que calcula um ponto de corte para cada sequência. A sequências são divididas no seu ponto de corte, sendo que a primeira metade das mesmas é colocada em um grupo e a outra é colocada em outro grupo. Se o comprimento de uma das subsequências do grupo é maior que um valor limite pré-estabelecido, o Algoritmo Genético é novamente aplicado para calcular pontos de corte para as mesmas. Esse processo é repetido até que todas as subsequências de todos os grupos tenham comprimento menor que o valor limite. Na fase 2, cada grupo de subsequências gerado na fase 1 é alinhado pelas técnicas de programação dinâmica (DP) [30]. Então, na fase 3, todos as subsequências alinhadas nos grupos são unidas formando o alinhamento final. O funcionamento de algoritmo é ilustrado no exemplo da Figura 4.12.

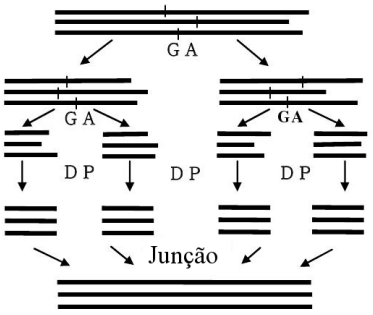

Figura 4.12: *Exemplo do funcionamento do algoritmo DC-GA [44].*

O Algoritmo Genético utilizado para encontrar os pontos de corte segue a estrutura do AG típico, ilustrado na seção 2 na Figura 2.1. A população do AG é formada por diferentes combinações de pontos de corte, um ponto de corte para cada sequência do conjunto de sequências a serem alinhadas. Uma combinação é chamada de cromossomo. Cada cromossomo da população é avaliado por uma fórmula que avalia os pontos de corte de cada combinação de pares de sequências, considerando a similaridade das mesmas. Depois da população ter sido avaliada, seus elementos são ordenados de acordo com seu valor, então eles são selecionados pelo método Roda Roleta, o mesmo método utilizado pelo algoritmo SAGA (seção 3.5.2). Dois cromossomos são escolhidos pelo operador de seleção. Em seguida, é gerado um valor aleatório entre 0 (zero) e 1, caso ele seja maior ou igual que o valor da taxa de *crossover*, o *crossover* de 1 ponto ((seção 2.4)) é aplicado às sequências. Os filhos gerados na etapa de *crossover* serão submetidos ao operador de mutação. O operador seleciona um ponto de corte do cromossomo de forma aleatória. Depois, ele gera um valor aleatório entre 0 (zero) e 1. Se esse valor for maior ou igual ao da taxa de mutação, o operador de mutação é aplicado ao cromossomo, substituindo o valor de ponto de corte selecionado por um valor aleatório. O operador seleção é novamente executado seguido dos operadores de *crossover* e mutação, até que o número de filhos gerados sejam iguais ao número de elementos na população pai. Esse processo é repetido até que o critério de parada seja satisfeito.

4.4.1 Resultados

O método foi implementado utilizando *JCreator* versão 2.5 [2] em um computador Pentium 4. As sequências de DNA utilizadas para alinhamento foram retiradas dos artigos [24, 62] e depois comparados os resultados dos alinhamentos obtidos com o método de Hirschberg [36]. Os parâmetros do utilizados no algoritmos foram o valor limite do tamanho da sequência (1/8 do comprimento da menor sequência), o número de gerações (100), o tamanho da população (120), a taxa de *crossover* $(0,7)$ e a a taxa de mutação $(0,01)$.

O resultados obtidos pelo método proposto foram ligeiramente melhores do que os resultados obtido pelo método de Hirschberg [36].

Pelo fato de se tratar de um método de divisão e conquista, o método proposto tem a vantagem de apresentar consumo de recursos de memória reduzido, possibilitando o trabalho com sequências longas e um grande número de sequências.

4.5 Sumário dos Resultados

Em seguida, são relatados os resultados mais relevantes obtidos pelos algoritmos.

Os resultados obtidos pelo iPGA (seção 4.1) mostram que o algoritmo pode obter alinhamentos com qualidade superior ao CLUSTAL-W [26] e ao AGS. Eles também evidenciam que a mudança nos intervalos de migração e no tamanho da população migrada afetam diretamente o resultado final, sendo os melhores resultados obtidos para intervalos mais longos de migração e com poucos elementos migrados.

Os resultados obtidos pelo PHGA (seção 4.2) mostram que o algoritmo pode obter alinhamentos com boa qualidade (comparáveis aos resultados do CLUSTAL-W e aos do PRRP). Também foi mostrado que os resultados melhoraram depois da mudança da função de aptidão do algoritmo para uma mais pesada. O algoritmo conseguiu escalar bem a medida que os problemas a serem processados eram maiores, de forma que o tempo gasto não foi alto demais (teste mais pesado foi para 24 sequências de comprimento máximo 473, gastou o tempo $6.100,92$ segundos). Ele utiliza muito menos memória e executa em muito menos tempo que as estratégias SAGA, OMA e MSA. Ele teve um *speed-up* de 3 para uma máquina de 4 processadores.

Os resultados obtidos pelo QGA (seção 4.3) mostram que o algoritmo obteve alinhamentos de alta qualidade, com resultados superiores ou próximos aos das estratégias CLUSTAL, DIALIGN e T-COFFEE. Um dos motivos da qualidade do alinhamento se deve ao uso função de aptidão COFFEE [34]. Em comparação com AGs clássicos, observa-se que estruturas de dados e as formas utilizadas para representar os alinhamentos reduziram eficientemente o tamanho populacional e o número de iterações para se obter uma boa solução.

O método DC-GA proposto na seção 4.4 é o único que não utiliza a estratégia do Algoritmo Genético para solucionar diretamente o problema do Alinhamento Múltiplo de Sequências. Ele a utiliza para encontrar pontos de corte no alinhamento e assim alinhar as partes de forma separada. É interessante ressaltar que nesse algoritmo, mesmo que cada parte do alinhamento tenha sido alinhada de forma separada, o algoritmo conseguiu obter resultados ligeiramente melhores que os do método de Hirschberg [36] para um dada base. Também, é importante destacar que pelo fato de se tratar de um método de divisão é conquista, o método proposto apresenta consumo reduzido de memória.

4.6 Quadro Comparativo

A Tabela 4.1 mostra algumas características dos algoritmos iPGA, PHGA, QGA e DC-GA, de forma a facilitar a visualização das diferenças entre os algorit-

mos. Na coluna "Algoritmo" é colocado o nome do algoritmo e a referência aos artigos que o descrevem, a coluna "Paralelo" tem como resposta sim se o algoritmo for paralelo e não caso contrário. A coluna "Estratégia Paralela" tem o nome da estratégia de paralelização utilizada no algoritmo. As quatro colunas seguintes "Inicialização", "*Crossover*", "Mutação" e "Avaliação" são correspondentes aos métodos utilizados nessas etapas do Algoritmo Genético.

Dos 4 algoritmos apresentados na Tabela 4.1, 3 utilizam o algoritmo genético para resolver diretamente o Problema do Alinhamento Múltiplo de Sequências (iPGA, PHGA e QGA) e o outro usa o AG em somente uma das etapas da resolução do problema(DC-GA). Dos 3 algoritmos que utilizam o AG diretamente para o MSA, dois são paralelos e utilizam a estratégia de ilha para paralelização. O algoritmo que não utiliza o AG diretamente para resolver o MSA também não utiliza nenhuma estratégia de paralelização para o AG, no entanto, o algoritmo como todo é paralelizado pela Técnica de Divisão e Conquista. Na coluna "Inicialização", observa-se que cada algoritmo utiliza uma estrutura de dados diferente. Dos algoritmo que explicitam a forma utilizada para gerar a população inicial, o iPGA inicializa de forma aleatória e o PHGA utiliza uma técnica modificada do alinhamento progressivo para a inicialização. Todos os algoritmos utilizam um operador de *crossover* semelhante ao *crossover* de 1 ponto (seção 2.4.1, Figura 2.3), no entanto, algumas das estratégias tiveram que adaptá-lo para à estrutura de dados. O *crossover* uniforme é utilizado pelo iPGA e pelo QGA. O iPGA implementa alguns dos operadores de mutação do SAGA (seção 3.5.2), enquanto as outras estratégias, implementam um operador específico para a estrutura de dados. As funções de aptidão utilizadas são diversas, a função WSP e COFFEE são as que obtém melhores resultados, o PHGA alterou sua função de aptidão SP para WSP no meio dos testes para melhorar os resultados.

Algoritmo	**Para-lelo**	**Estratégia Paralela**	**Inicialização**	***Crossover***	**Mutação**	**Avaliação**
iPGA [58, 57]	sim	ilha	Alinhamentos gerados de forma aleatória	uniforme, 1 ponto (seção 3.5.2)	inserção de espaço em branco, deriva de blocos, pesquisa por blocos, (seção 3.5.2)	cálculo WSP (seção 3.4) PAM250 [53] *natural affine gap* [23]
PHGA [64, 65]	sim	ilha	Alinhamentos gerados por versão modificada do alinhamento progressivo	Operador específico para a estrutura de dados (seção 4.2)	Operador específico para a estrutura de dados (seção 4.2)	cálculo SP (seção 3.4) PAM250 [53] *quasi-nature* [23]
QGA [49]	não	-	cria 4 matrizes quânticas	uniforme, de 1 ponto adaptado para a estrutura de dados (seção 4.3)	Altera *qbits* dentro do cromossomo (seção 4.3)	COFFEE [34]
DC-GA [44]	sim	Divisão e conquita [36]	conjunto de pontos corte	de 1 ponto (seção 2.4)	DC-GA 1 elemento dentro do cromossomo (seção 4.4)	função específica (seção 4.4)

Tabela 4.1: *Quadro comparativo.*

Capítulo 5

Projeto da Estratégia Paralela Multi-ilha

5.1 Escolhas de Projeto

Como já abordado na seção 3, o Alinhamento Múltiplo de Sequências (MSA) é realizado milhares de vezes ao dia por cientistas, a fim de encontrar regiões de semelhanças entre as sequências, de forma que se possa inferir propriedades genéticas dessas regiões. No entanto, o MSA é um problema NP-completo [40], logo, heurísticas são normalmente utilizadas para solucionar esse problema.

Conforme descrito na seção 3.3, existem diversas abordagens heurísticas para o MSA, que obtem soluções boas em tempo aceitável. Atualmente, uma das principais estratégias para solucionar o problema do MSA é a estratégia progressiva [20], cujos representantes significativos são o CLUSTAL-W [26] e o T-COFFEE [11]. Essa estratégia é relativamente rápida. No entanto, a maior desvantagem dela é que depois que a sequência foi alinhada não há como alterar o alinhamento, mesmo que sejam identificados conflitos com as sequências adicionadas posteriormente [51].

Outra alternativa são as abordagens iterativas. Os métodos iterativos utilizam um algoritmo que produz um alinhamento e em seguida esse alinhamento é melhorado por meio de sucessivas iterações. Nestes métodos, incluem-se as abordagens Modelos Escondidos de Markov (*hidden markov models* - HMM) [21, 39], Algoritmos Genéticos [27], algoritmos evolucionários (evolutionary programming) [25] e outros. Nessa abordagem, é reduzido o problema da estratégia progressiva, pois ela permite a correção do alinhamento nas diversas iterações. No entanto, os métodos nessa abordagem, em geral, consomem mais tempo que os da estratégia progressiva [51].

Como pode ser observado, existem diversas alternativas para resolver o mesmo problema. No estudo dos Algoritmos Genéticos (Capítulo 2) e AGs para MSA foi constatado que tais algoritmos são capazes de encontrar soluções muito boas, bastante próximas das soluções propostas manualmente pelos biólogos [54], em tempo razoável que são tão boas ou melhores que o CLUSTAL-W [43, 33, 51].

No Capítulo 4, foram estudadas diferentes estratégias do uso de Algoritmos Genéticos no MSA, especialmente estratégias paralelas. Dentre os quatro algo-

ritmos estudados, o iPGA (seção 4.1), o PHGA (seção 4.2) e o QGA (seção 4.3) utilizam o AG para solucionar diretamente o problema do alinhamento. Dentre eles, dois são paralelos (iPGA e PHGA) e um sequencial (QGA). Os dois algoritmos paralelos utilizam a estratégia de ilha para paralelização e obtiveram bons resultados. Geralmente, a estratégia mestre/escravo (seção 2.6) com população única não é utilizada nesse caso, pois a paralelização dentro do *loop* do AG provocaria um grande número de mensagens entre os nodos, o que retardaria a computação. Em um algoritmo como o SAGA (seção 3.5.2), o método de paralelização de ilhas é uma alternativa bastante viável. Em [31], são mostrados casos em que a estratégia de ilha obtém resultados melhores que a estratégia sequencial.

Na seção 2.7, foi visto que, em [28], foi proposta uma estratégia de AG paralelo para melhorar o desempenho do AGS (Algoritmo Genético Sequencial) na resolução do problema da gerência de recursos hídricos. A estratégia combinava as estratégias de paralelização mestre/escravo e de ilha (seção 2.6). No modelo proposto em [28], existem três ilhas onde, em cada uma, é executado um algoritmo que utiliza a estratégia mestre/escravo para paralelizar a etapa de avaliação do problema. Em duas das três ilhas (Ilhas de Baixa Resolução) é implementada uma função de avaliação simples e rápida e na outra ilha (Ilha de Alta Resolução) é implementada uma função de avaliação complexa que obtém resultados de maior qualidade que a primeira. As ilhas trocam elementos entre si em intervalos pré-definidos. Segundo os autores, o modelo apresentado direcionaria o algoritmo para soluções melhores em um tempo mais curto [28].

Diante da possibilidades de melhoria de resultado com o uso do modelo descrito em [28], foi pensado em um modelo parecido para o problema do MSA. O modelo utiliza a mesma estratégia de ilhas proposta em [28], onde existem uma Ilha de Alta Resolução e duas Ilhas de Baixa Resolução trocando elementos entre si em intervalos pré-definidos. Diferente do modelo proposto, dentro dessas ilhas, é implementado um AG paralelizado pela estratégia de ilha. Na Ilha de Baixa Resolução é utilizado um Algoritmo Genético que divide cada alinhamento em partes e, em seguida, evolui cada conjunto de partes de forma separada. O algoritmo da Ilha de Baixa Resolução melhora o alinhamento de cada parte de forma mais precisa, pois a maior parte dos operadores de *crossover* e mutação trabalham com blocos do alinhamento, e esses operadores são aplicados à partes menores. Quando os elementos são trocados entre as ilhas (Ilha de Alta Resolução e Ilhas de Baixa Resolução), a Ilha de Alta Resolução, que executa um Algoritmo Genético convencional de ilha, recebe os alinhamento da Ilha de Baixa Resolução e utiliza, no processo evolutivo, algumas das partes boas dos alinhamentos nas novas soluções, gerando indivíduos mais aptos de forma mais rápida.

5.2 Visão Geral

Tendo por base o algoritmo exposto na seção 5.1, o algoritmo proposto é constituído de três ilhas, uma Ilha de Alta Resolução (SIAR) e outras duas Ilhas de Baixa Resolução (SIBR), conforme o modelo ilustrado na Figura 5.1. Essas ilhas serão chamadas de Super Ilhas para diferenciar das outras ilhas presentes no algoritmo.

Em cada Super Ilha, será implementado um Algoritmo Genético que utiliza o modelo de paralelização de ilha para resolver o problema do Alinhamento Múltiplo de Sequências.

Na SIAR (Figura 5.1), é implementado um Módulo de Alta Resolução (MAR) que é responsável por gerar a população inicial composta por alinhamentos múltiplos, dividir a mesma em subgrupos e enviar um subgrupo para uma ilha, de modo que cada ilha tenha um subgrupo da população. Essas ilhas serão chamadas de Micro Ilhas. Cada Micro Ilha executa um Algoritmo Genético Base (AGB), detalhado na seção 5.5, que evolui a população recebida. Em intervalos pré-definidos, chamados de Intervalos de Migração Interna, cada Micro Ilha interrompe a execução do AGB, envia os i indivíduos mais aptos da população para a Micro Ilha vizinha da direita, recebe os i indivíduos mais aptos da vizinha da esquerda, insere na população os indivíduos recebidos, substituindo os indivíduos enviados pelos recebidos e dá continuidade ao processo de evolução da população. Em intervalos maiores também pré-definidos, chamados de intervalos de Migração Externa, cada Micro Ilha interrompe a execução do AGB, envia os i indivíduos mais aptos da população para o MAR. O MAR aguardará receber os indivíduos de cada Micro Ilha, selecionando os indivíduos mais aptos e os enviando às SIBR. Em seguida, ele aguardará o recebimento dos indivíduos das SIBR e envia os mais aptos para as Micro Ilhas. Esse processo é repetido até que o AGB satisfaça o critério de parada, ou seja, se já tiver atingido o número máximo de gerações. Depois disso, o indivíduo mais apto da população é o resultado final.

A SIBR (Figura 5.1) possui muitas semelhanças com a SIAR, inclusive o algoritmo executado nas Micro Ilhas é o mesmo, diferenciando-se somente no intervalo de Migração Interna, onde os elementos não são enviados e recebidos dos vizinhos, mas do Módulo de Baixa Resolução (MBR). O MBR é o principal diferencial entre as implementações dos tipos de Super Ilha. Ele também gera a população inicial, mas ao invés de dividir a população em subgrupos, ele corta cada elemento i da população em x partes $(pa_{i1}, pa_{i2}, ..., pa_{ix})$, sendo $parte_1$ o conjunto das primeiras partes $(pa_{11}, pa_{21}, ..., pa_{n1})$ e $parte_x$ o conjunto das últimas partes. O MBR envia $parte_1$ para a Micro Ilha 1, $parte_2$ para a Micro Ilha 2 e assim por diante, de modo que cada Micro Ilha tenha um conjunto $parte_j$. Nos intervalos de Migração Interna, as Micro Ilhas enviam sua população inteira para o MBR. O MBR une as partes $(p_1, p_2, ..., p_x)$, de forma que os alinhamentos gerados sejam válidos, executa sobre os alinhamentos um operador, seleciona os elementos mais aptos, parte novamente os alinhamentos selecionados, mas em pontos diferentes e envia os conjuntos de partes para as Micro Ilhas que darão continuidade à execução do AGB. Esse processo será explicado com mais detalhes na seção 5.4. No intervalo de Migração Externa, o MBR recebe as populações de cada Micro Ilha, junta as partes, aplica o operador e envia os elementos mais aptos para a outra SIBR e para a SIAR (Figura 5.1). Em seguida, o MBR aguarda o recebimento dos elementos que as Super Ilhas enviarão. Depois de recebidos, ele seleciona os elementos mais aptos, os divide em partes e os envia para as Micro Ilhas. Esse processo é repetido até que o AGB atingido o número máximo de gerações. Na última geração, as Micro Ilhas enviam a população para o MBR e terminam a execução. O MBR envia os elementos mais aptos para a SIAR e, por sua vez, termina a execução.

O modelo proposto pode ter variantes. Nesse trabalho, serão abordadas duas implementações do modelo que exploram as Migrações Externas de forma diferenciada (Capítulo 6). Os SIAR, SIBR e AGB serão descritos com mais detalhes nas seções 5.3, 5.4 e 5.5, respectivamente.

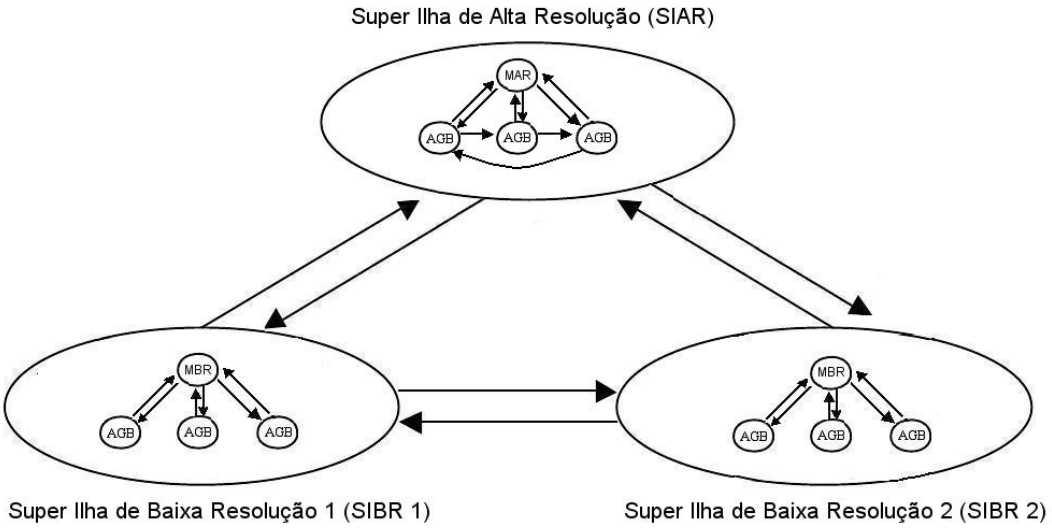

Figura 5.1: *Adaptação da estratégia paralela descrita em [28].*

5.3 Super Ilha de Alta Resolução (SIAR)

Nessa seção, será descrita a SIAR e cada etapa descrita é seguida pelo número da linha do algoritmo ilustrado na Figura 5.2 que corresponde àquela etapa.

Primeiramente, o MAR gera a população inicial por um método aleatório (Figura 5.2, linha 1), de maneira que a população não possua elementos repetidos. A população inicial é, então, dividida em subpopulações que tem o mesmo tamanho e distribuída entre as n Micro Ilhas (linha 2). Em cada Micro Ilha, é executado o AGB (seção 5.5). Se o critério de parada já tiver sido satisfeito (linha 3), o algoritmo recebe as populações que estão nas Micro Ilhas (linha 9), finaliza o algoritmo (linha 10) e o melhor indivíduo das populações recebidas das Micro Ilhas é o resultado final. Se não, o MAR entra no *loop* (linha 3). Dentro do *loop*, ele recebe o total de i indivíduos (os mais aptos) de todas as Micro Ilhas (linha 4), que são imediatamente enviados para as outras Super Ilhas (linha 5). Em seguida, ele fica bloqueado até receber, de cada Super Ilha, i indivíduos novos (linha 6) que serão enviados para as Micro Ilhas (linha 7) para compor a população das mesmas. A esse processo chamamos de Migração Externa, pois as Super Ilhas trocam elementos entre si. O *loop* se repete até que o critério de parada seja atingido.

Módulo de Alta Resolução (MAR)

1. **Gerar** população inicial
2. **Distribuir** população inicial entre as Micro Ilhas
3. **Enquanto** critério de parada não satisfeito **faça**
4. **Receber** i indivíduos de todas as Micro Ilhas
5. **Migrar** i indivíduos para cada Super Ilha
6. **Receber** i indivíduos de cada Super Ilha
7. **Distribuir** i indivíduos entre as Micro Ilhas
8. **Fim Enquanto**
9. **Receber** i indivíduos de todas as Micro Ilhas
10. **Finalizar Algoritmo**

Figura 5.2: *Algoritmo de Alta Resolução.*

A Figura 5.3 ilustra o funcionamento do algoritmo na Figura 5.2, sendo que os números na Figura 5.3 correspondem aos das linhas do algoritmo apresentado na Figura 5.2.

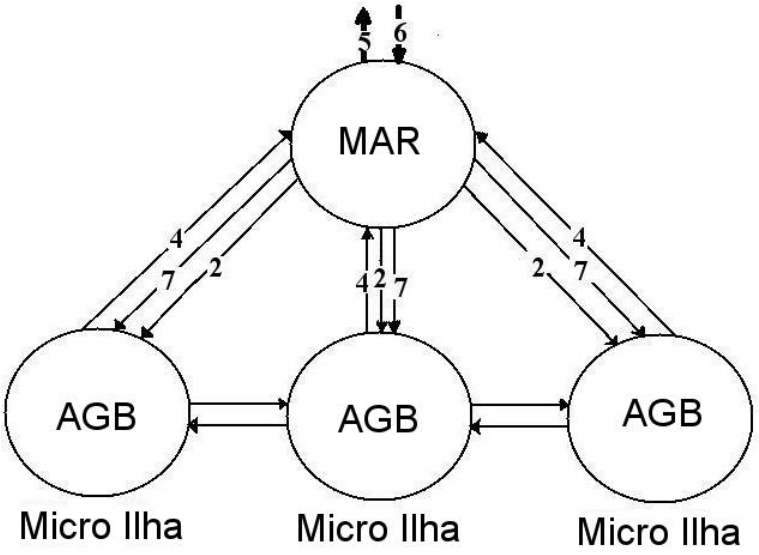

Figura 5.3: *Ilustração do algoritmo do Módulo de Alta Resolução.*

5.3.1 População Inicial

A população inicial é gerada a partir do grupo de sequências que serão alinhadas. Primeiro, é criado um alinhamento base com as sequências com espaços em branco terminais (seção 3.5.2). As sequências são colocadas uma sobre a outra

de acordo com a ordem de entrada, a primeira no topo, a segunda abaixo da primeira e assim por diante. Depois, é aplicado ao alinhamento base um operador que insere espaços em brancos aleatoriamente dentro do alinhamento, gerando um novo alinhamento. Em seguida, é medida a aptidão do alinhamento gerado com a função SP (seção 2.4). Se ele possuir a mesma aptidão de algum dos alinhamentos gerados anteriormente, o alinhamento é eliminado, com a finalidade de eliminar elementos repetidos da população. O processo é repetido até que a população seja gerada em sua totalidade. O tamanho da população é definido em um arquivo de configuração.

Operador *Gap Generator* (GG)

O operador *Gap Generator* (GG) calcula o número de espaços em branco a serem gerados em cada sequência do alinhamento, baseado no comprimento da maior sequência do alinhamento e no parâmetro que determina a porcentagem de aumento do comprimento do alinhamento. De posse do número de espaços em branco que vão ser colocados na linha do alinhamento, o algoritmo gera um número aleatório entre um e o número de espaços em branco e outro entre zero e o comprimento da maior sequência somado à porcentagem, para definir, respectivamente, o número de espaços em branco que vão ser gerados e a posição de início da inserção. Esse processo é repetido até que todos os espaços em branco sejam inseridos em todas as sequências.

5.3.2 Divisão da População

A população é dividida igualmente entre as Micro Ilhas. Caso o número de Micro Ilhas não seja múltiplo do tamanho da população, as primeiras ilhas ficam com mais indivíduos. A população é dividida considerando-se a aptidão dos alinhamentos, de modo que cada Micro Ilha receba uma quantidade parecida de indivíduos muito aptos, médio aptos e pouco aptos.

5.3.3 Critério de Parada

O critério de parada do MAR depende do número de migrações do AGB, pois ele tem que executar exatamente o número de intervalos de Migração Externa que o AGB identifica (Figura 5.1).

5.4 Super Ilha de Baixa Resolução (SIBR)

Da mesma forma que na seção 5.3 foi descrita a SIAR, nessa seção será descrita a SIBR. Assim como no SIAR, cada etapa descrita é seguida pelo número da linha do algoritmo ilustrado na Figura 5.4 que corresponde àquela etapa. A Figura 5.5 ilustra o funcionamento do algoritmo na Figura 5.4, sendo que os números na Figura 5.5 correspondem aos das linhas do algoritmo apresentado na Figura 5.4.

Primeiramente, o MBR gera a população inicial por um método aleatório (Figura 5.4,linha 1), de maneira que a população gerada não possua elementos

repetidos. Se o critério de parada tiver sido satisfeito o algoritmo finaliza. Se não, ele verifica se o intervalo de Migração Externa foi atingido (linha 3). Se sim, ele envia os i indivíduos mais aptos da população (linha 4) para cada Super Ilha, recebe i indivíduos de cada Super Ilha (linha 5), seleciona os i mais aptos (linha 6) e os insere no lugar dos indivíduos enviados (linha 7). Depois, os indivíduos da população são divididos em n partes (linha 9), onde n é igual o número de Micro Ilhas. O conjunto das primeiras partes são enviados para a Micro Ilha 1, os da m-ésimas partes são enviados para a Micro Ilha m e assim por diante (linha 10). Cada Micro Ilha executará o AGB (seção 5.5). Em seguida, o algoritmo recebe os grupos de partes das Micro Ilhas (linha 11) e monta novos alinhamentos com as partes (linha 12). Os alinhamentos são avaliados pelo cálculo SP descrito na seção 3.4 (linha 13) e os piores são eliminados (linha 14). O *loop* se repete até que o critério de parada seja atingido.

Módulo de Baixa Resolução (MBR)

```
1.  Gerar população inicial
2.  Enquanto critério de parada não satisfeito faça
3.      Se Intervalo de Migração Externa
4.          Migrar i indivíduos para cada Super Ilha
5.          Receber i indivíduos de cada Super Ilha
6.          Selecionar i indivíduos mais aptos
7.          Inserir i indivíduos na população
8.      Fim Se
9.      Dividir os indivíduos em partes
10.     Migrar conjuntos de partes para as Micro Ilhas
11.     Receber conjunto de partes de cada Micro Ilha
12.     Juntar as partes
13.     Avaliar os indivíduos
14.     Eliminar indivíduos menos aptos
15. Fim Enquanto
```

Figura 5.4: *Algoritmo de Baixa Resolução.*

Na Figura 5.5, temos uma população inicial composta por 3 alinhamentos múltiplos (Figura 5.5 (1)). Os 3 alinhamentos múltiplos são quebrados em 3 partes, que correspondem ao número de Micro Ilhas (Figura 5.5 (7), parte superior). As 3 partes são enviadas para as 3 Micro Ilhas, sendo que as partes iniciais vão para a Micro Ilha 1, as partes do meio vão para a Micro Ilha 2 e as partes finais vão para a Micro Ilha 3 (Figura 5.5 (8)). As Micro Ilhas executam o Algoritmo Genético Base (AGB), detalhado na seção 5.5, e retornam os melhores indivíduos (Figura 5.5 (9)). As partes são juntadas, de maneira a formar um alinhamento

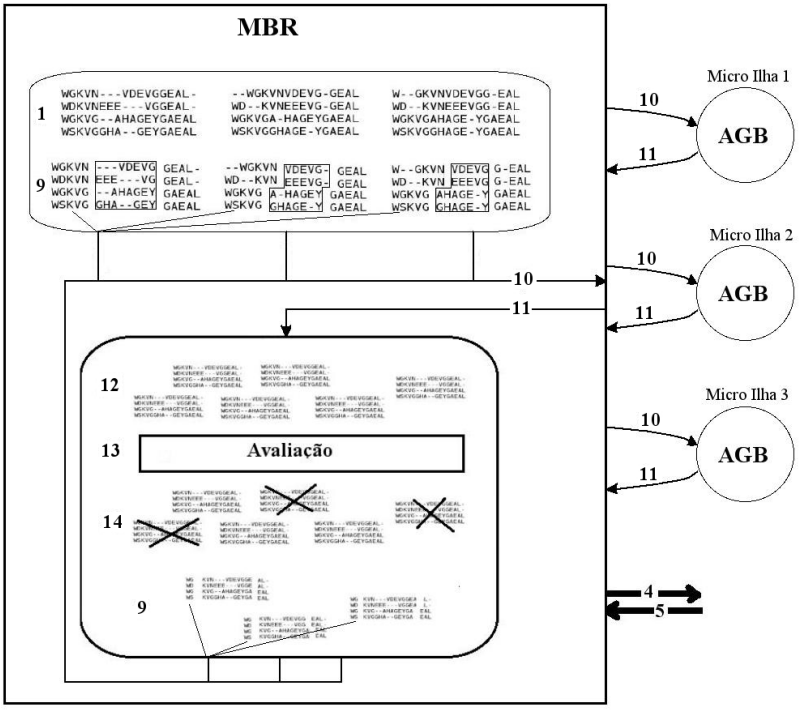

Figura 5.5: *Ilustração do algoritmo de Baixa Resolução. Os números na figura correspondem aos das linhas do algoritmo apresentado na Figura 5.4*

completo (Figura 5.5 (10)). Os alinhamentos são avaliados (Figura 5.5 (11)) e os piores são descartados (Figura 5.5 (12)). A população restante é então dividida em partes (Figura 5.5 (7), parte inferior) e o processo é repetido novamente. A cada intervalo de Migração Externa, os melhores indivíduos são enviados para as Super Ilhas (Figura 5.5 (4)) e novos indivíduos são recebidos de cada Super Ilha (Figura 5.5 (5)).

5.4.1 População Inicial

A população inicial é gerada a partir do mesmo método utilizado pela SIAR (seção 5.3).

5.4.2 Divisão da População

Cada indivíduo da população é inspirado na etapa de corte do *crossover* de 1 ponto do SAGA (seção 3.5.2.4). O número de caracteres da primeira sequência de um dos alinhamentos é contado e esse número é dividido pelo número de processadores, obtendo-se um valor x. São feitos cortes no primeiro indivíduo a cada x caracteres da primeira sequência. Depois, os outros indivíduos são partidos de modo a i-ésima parte deles tenha os mesmos caracteres da i-ésima parte do primeiro indivíduo. Na etapa seguinte de corte, o ponto de corte da primeira sequência é calculado da seguinte maneira: $ponto\_de\_corte = ponto\_anterior - numero\_caracteres/(numero\_processos * 2)$, onde o $ponto\_anterior$ é o ponto de corte prévio, $numero\_caracteres$ é o número de caracteres da primeira sequência

do alinhamento e *numero_processos* é o número de AGBs de Baixa Resolução (Micro Ilhas). Dessa maneira, o alinhamento cortado em partes diferentes, a cada vez.

O método implementado para juntar as partes dos indivíduos previamente divididos (linha 10) foi feito de forma bem simples. O método reconecta as partes, que antes tinham sido divididas do indivíduo, formando como resultado alinhamentos válidos. A fim de remanejar os caracteres dentro dos blocos de espaços em branco, que são criados com a junção das partes do alinhamento, são aplicados sobre os alinhamentos os operadores GCS ou CGCS (que serão detalhados na seção 5.4.4). Ao aplicar esses operadores, alguns caracteres são migrados para direita ou para a esquerda do alinhamento e quando esse alinhamento é partido novamente esses caracteres estarão em uma nova parte. Isso reduz a probabilidade de ocorrência de ótimos locais, que poderiam acontecer, já que cada parte está sendo evoluída de forma separada.

5.4.3 Critério de Parada

O critério de parada do MBR depende do número de migrações do AGB, pois ele tem que executar exatamente o número de intervalos de Migração Interna e Externa que o AGB identifica (Figura 5.1).

5.4.4 Operadores de Mutação Propostos

O operadores de mutação propostos nessa dissertação movem caracteres de uma coluna para outra sobre subsequências formadas por espaços, com o objetivo de formar alinhamentos mais aptos. Os operadores utilizam o cálculo SP (seção 3.4) para decidir se os caracteres de serão movidos, formando sempre alinhamentos iguais ou mais aptos que os de entrada. Eles podem ser implementados para mover os caracteres para direita ou para esquerda, a seguir eles foram descritos movendo os caracteres para esquerda. Também, eles podem ser adaptados para o cálculo WSP (seção 3.4) com facilidade.

Operador *Gap Char Shifter* (GCS)

O operador GCS tem o objetivo caracteres sobre subsequências de espaços em branco, com o objetivo de tornar o alinhamento mais apto. O GCS percorre cada coluna do alinhamento até encontrar um espaço em branco (Figura 5.6.a). Quando um espaço em branco é encontrado, o operador procura nas colunas seguintes um caractere que esteja na mesma linha do espaço em branco (Figura 5.6.b). Se o GCS o encontrar, o operador calcula a aptidão das colunas pelo cálculo SP (seção 3.4). Depois, ele calcula novamente a aptidão das colunas utilizando o mesmo cálculo, no entanto, agora ele supõe que os elementos foram trocados, isto é, o espaço em branco está na coluna onde estava o caractere e vice-versa. Se o valores de aptidão das colunas com os elementos trocados for maior que o anterior, o espaço em branco é colocado no lugar do caractere e o caractere no lugar do espaço em branco (Figura 5.6.c). Esse processo é repetido para cada espaço em branco do alinhamento.

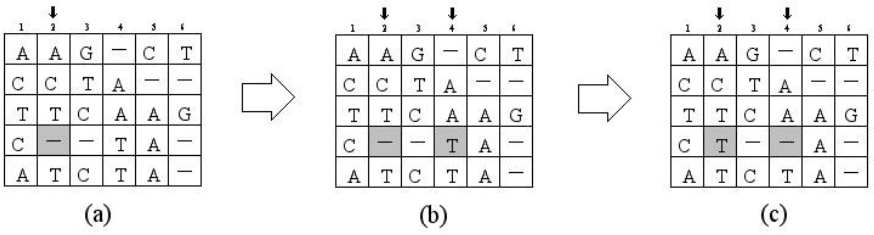

Figura 5.6: *Exemplo do funcionamento do operador GCS.*

Operador *Column Gap Char Shifter* (CGCS)

O operador CGCS tem o objetivo de mover conjuntos de caracteres em blocos de espaços em branco, com o objetivo de tornar o alinhamento mais apto. O CGCS percorre cada coluna i do alinhamento procurando espaços em branco. Quando um ou mais espaços em branco são encontrados em uma mesma coluna, ele mapeia todos os espaços em branco da mesma (Figura 5.7.a). Em seguida, ele procura, nas colunas seguintes, algum caractere que esteja em uma das linhas onde existe espaço em branco na coluna i (Figura 5.7.b). Quando ele encontra a coluna j, o operador mapeia os caracteres da coluna j que estão na mesma linha dos espaços em branco da coluna i e calcula todas as combinações possíveis de troca de caracteres por espaço em branco (Figura 5.7.c). Depois, ele calcula a aptidão de todas as possíveis combinações. A combinação mais apta é selecionada e os caracteres e espaços em branco são permutados de acordo com a combinação (Figura 5.7.d). O processo é repetido para os espaços em branco na coluna que ainda não foram testados a possibilidade de permuta (Figura 5.7.e). Depois que todos os espaços em branco da coluna foram testados, o operador vai para a próxima coluna $i + 1$ executar o mesmo processo (Figura 5.7.f).

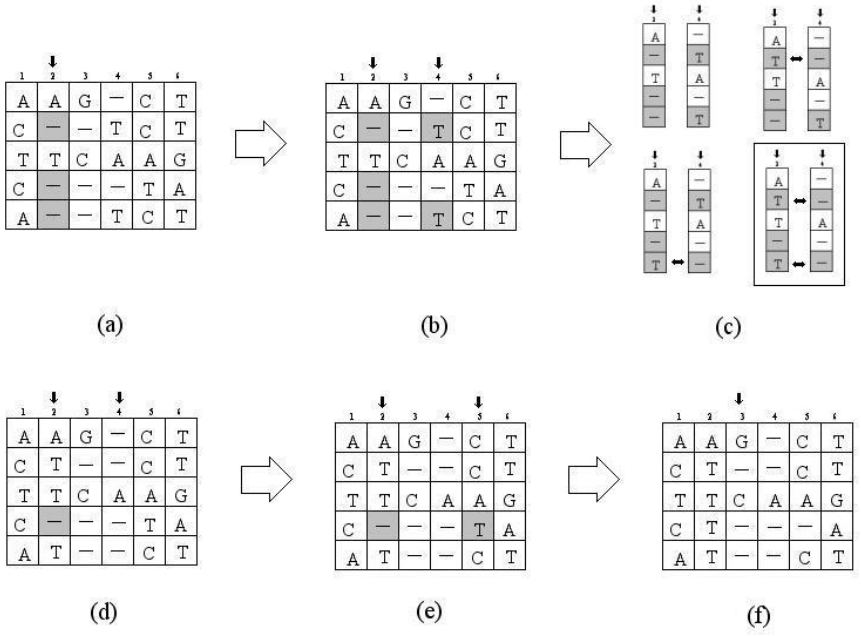

Figura 5.7: *Exemplo do funcionamento do operador CGCS.*

5.5 Algoritmo Genético Base (AGB)

Nessa seção, será descrito o Algoritmo Genético implementado nas Micro Ilhas (Figura 5.1) chamado de Algoritmo Genético Base (AGB), ilustrado na Figura 5.8). A concepção do AGB foi inspirada no SAGA (seção 3.5.2). No entanto, pela própria estrutura Multi-ilha na qual o AGB está inserido foram feitas diversas modificações. A principal mudança é a inserção de uma etapa de migração (linhas 6 e 7). Além disso, alguns operadores foram implementados de forma simplificada e novos operadores foram propostos.

Primeiramente, o AGB recebe a população inicial G_1 (Figura 5.8, linha 1) e avalia a mesma com o cálculo SP (linha 3). Se o critério de parada tiver sido atingido (linha 4), o algoritmo pára e envia a população G_n (linha 18) para o receptor designado. Se não, ele entra no *loop* (linha 5). Dentro do *loop*, ele, primeiro, verifica se o intervalo de migração foi atingido (linha 5). Se sim, ele envia os i indivíduos mais aptos (linha 6) para um outro nó previamente determinado e os substitui pelos novos i indivíduos que ele receberá (linha 7). Depois, é iniciada a etapa de geração de filhos. Primeiro, são selecionados os 50% indivíduos mais aptos da população G_n que comporão a população da próxima geração G_{n+1} (linha 9). Com isso, são maiores as possibilidades da população convergir para uma solução boa e também da convergência ser mais rápida. Depois, o AGB entra em um *loop* para gerar o resto dos indivíduos que comporão a nova população (linha 10). Dentro do *loop*, são selecionados na população G_n, os indivíduos que serão utilizados para gerar filhos pelo método da Roda Roleta (linha 12). Em seguida, o AGB seleciona operadores de *crossover* e *mutação* que serão aplicados

Algoritmo Genético Base (AGB)

1. **Receber** G_1
2. $n = 1$
3. **Avaliar** população G_n
4. **Enquanto** critério de parada não satisfeito **faça**
5. **Se** $((n\ mod\ NUMERO\_GERACOES\_MIGRACAO) = 0)$
6. **Migrar** i indivíduos
7. **Receber** i indivíduos
8. **Fim Se**
9. **Selecionar** 50% mais aptos de G_n e colocar em G_{n+1}
10. **Enquanto** população não está completa **faça**
11. **Selecionar** pelo método Roda Roleta os indivíduos pais
12. **Selecionar** pelo método Roda Roleta o operador que será apliclado indivíduos pais para gerar os filhos
13. **Gerar** filhos
14. **Inserir** filhos em em G_{n+1}
15. **Fim Enquanto**
16. **Avaliar** população G_{n+1}
17. $n = n + 1$
18. **Fim Enquanto**
19. **Enviar** G_n

Figura 5.8: *Algoritmo Genético Base (AGB).*

a esses indivíduos, também pelo método da Roda Roleta (linha 12). Então, são gerados alguns dos filhos que comporão a outra metade da população G_{n+1} (linha 13). Se o filho gerado já existir na população, ele é eliminado e é gerado um novo filho. O *loop* é repetido até que a população esteja completa. Por fim, a nova geração (G_{n+1}) é avaliada (linha 16). O *loop* se repete até que o critério de parada seja atingido, isto é, o número de gerações seja igual ao definido no início da execução.

5.5.1 População Inicial

A etapa de geração da população inicial não é feita por esse algoritmo, já que ela é recebida ou do MAR ou do MBR.

5.5.2 Avaliação

O cálculo de aptidão implementado foi o SP (seção 3.4), com a função de custo de espaços em branco (*natural affine gap cost*) [64] (seção 3.2.1.1) e matriz de substituição PAM250 [53].

5.5.3 Geração de filhos

Os operadores implementados foram operadores de *crossover* de 1 e 2 pontos e operadores de mutação deriva de blocos, pesquisa por blocos, remover coluna

de espaços em branco e inserção de espaço em branco (seção 3.5.2). O operador inserção de espaço em branco foi implementado de forma simplificada: ao invés da inserção de espaços em branco ser feita com base em uma árvore filogenética, ela é feita de forma aleatória por um método semelhante ao descrito na seção 5.3.1). Também foram utilizados os operadores GCS e CGCS, propostos na seção 5.4.4.

5.6 Glossário

Glossário de termos utilizados nesse capítulo:

- Algoritmo Genético (AG)
- Algoritmo Genético Sequencial (AGS)
- Alinhamento Múltiplo de Sequências (MSA)
- Super Ilha de Alta Resolução (SIAR)
- Super Ilha de Baixa Resolução (SIBR)
- Módulo de Alta Resolução (MAR)
- Módulo de Baixa Resolução (MBR)
- Algoritmo Genético Base (AGB)
- Operador *Gap Generator* (GG)
- Operador *Gap Char Shifter* (GCS)
- Operador *Column Gap Char Shifter* (CGCS)

Capítulo 6

Resultados Experimentais

Nesse capítulo, será feita uma contextualização de onde e como foram realizados os testes da estratégia proposta na seção 2.7 e em seguida serão apresentados e analisados os resultados obtidos.

6.1 Ambiente Experimental

A implementação algoritmo de Injeção de Ilhas proposto foi feita em ANSI C, utilizando o compilador GCC 4.1.1-21, no sistema operacional GNU/Linux Kernel 2.6.18-6-686. Para a comunicação entre os processos foi utilizado o padrão MPI, implementado pela biblioteca MPICH 1.2.7.

Os testes foram realizados em dois *clusters*. Os primeiros testes foram feitos no Cluster-MP, que é composto por 8 máquinas, cada uma com dois processadores *Core* 2 *Duo Intel(R) Xeon(TM)* $3,00$ *GHz*, com 2 *MB* de memória *cache*, 2 *GB* de memória *RAM* e 80 *GB* de disco rígido. Os máquinas foram conectadas por uma rede *Gigabit Ethernet* não dedicada. O segundo *cluster* utilizado para os testes, foi o Cluster-IE, com 4 máquinas, cada uma com dois processadores *AMD Athlon(TM) MP 1900+* $1,60$ *GHz*, com 256 *KB* de memória *cache*, 1 *GB* de memória *RAM* e 40 *GB* de disco rígido. Para conexão das máquinas, foi utilizada uma rede dedicada *Gigabit Ethernet*.

Nos testes, foram usadas 5 abordagens de Algoritmos Genéticos para Alinhamento Múltiplo de Sequências: *Injeção de Ilhas one-way* (i1), *Injeção de Ilhas two-way* (i2), *Paralela 9 ilhas* (p9), *Paralela 3 ilhas* (p3) e *Sequencial* (seq). As duas primeiras estratégias são diferentes implementações do algoritmo descrito no capítulo 5, utilizando modelos distintos de Migração Externa. Na estratégia de *Injeção de Ilhas one-way* (i1), a Migração Externa consiste em enviar indivíduos das Super Ilhas de Baixa Resolução (SIBR) para a Super Ilha de Alta Resolução (SIAR) no intervalo de Migração Externa (Figura 6.1.a). Por outro lado, na estratégia de *Injeção de Ilhas two-way* (i2), a SIAR também envia indivíduos para as SIBR no intervalo de Migração Externa (Figura 6.1.b). As estratégias *Paralela 9 ilhas* (p9) e *Paralela 3 ilhas* (p3) são implementações do modelo de ilha (seção 2.6), utilizando respectivamente 9 e 3 ilhas (Figura 6.1.c e 6.1.d). Em cada uma das estratégias p9 e p3 existe um nó Coordenador (Coord) que cria uma população inicial de forma aleatória e divide essa população entre as ilhas. Cada

ilha executa o AGB (seção 5) até atingir o critério de parada, que é o número de gerações, e então retorna os melhores indivíduos para o nó Coord, que selecionará, dentre os indivíduos recebidos de todas as ilhas, o mais apto para ser a solução final. Em intervalos, chamados intervalos de Migração, as ilhas enviam seus melhores indivíduos para o vizinho da direita e aguardam o recebimento de indivíduos enviados pelo vizinho da esquerda para dar prosseguimento a execução. Esse processo pode ser bem demorado, especialmente para um grande número de ilhas. A estratégia *Sequencial* (seq) é a implementação do Algoritmo Genético Sequencial, sem nenhum mecanismo de paralelização (Figura 6.1.e).

Na Figura 6.1.e, a estratégia seq é ilustrada por meio de um círculo, que contém um outro círculo escrito AGB. A estratégia foi ilustrada dessa forma, para mostrar que dentro dela é executado o algoritmo AGB (seção 5). Será chamado de Nó AGB, cada nó que executar o AGB, podendo o nó ser ilha (i1, i2, p9 e p3) ou não (seq).

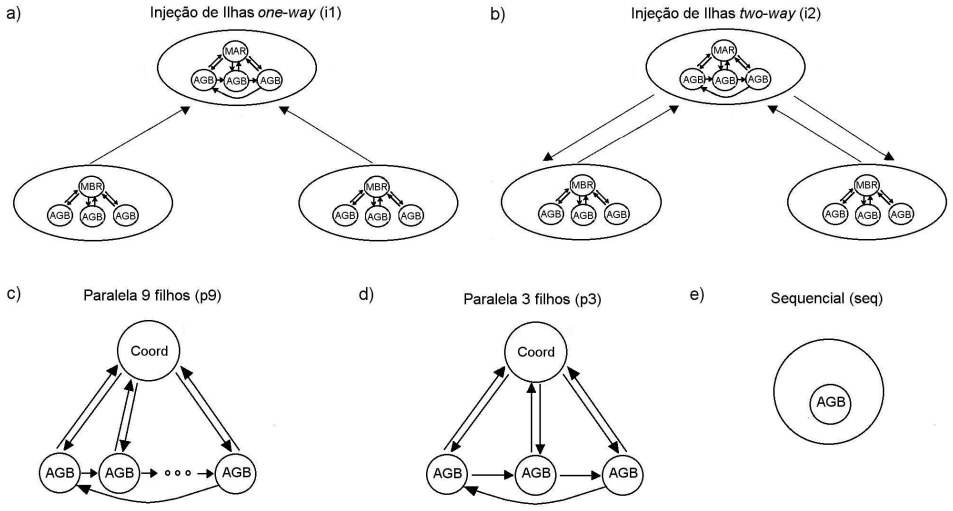

Figura 6.1: *Abordagens de AGs. a) Injeção de Ilhas one-way (i1). b) Injeção de Ilhas two-way (i2). c) Paralela 9 filhos (p9). d) Paralela 3 filhos (p3). e) Sequencial (seq).*

A Tabela 6.1 apresenta a disposição dos processos de cada abordagem (i2, i1, p9, p3 e seq) nas máquinas do Cluster-IE. O *cluster* possui um arquivo de configuração que indica como os processos serão alocados a determinada máquina à medida que os processos vão sendo criados. Então, se um arquivo de configuração diz que na máquina 1 serão alocados 2 processos e na máquina 2 será alocado 1 processo, os dois primeiros processos que forem criados, isto é, os que possuírem identificadores iguais a 0 e 1, são alocados à máquina 1 e o com identificador igual a 2 é alocado à máquina 2. Dentro de cada máquina o sistema operacional que decide como os processos serão alocados aos processadores.

No arquivo de configuração do Cluster-MP foi especificado somente as máquinas que seriam parte do *cluster*, não sendo especificado a distribuição dos processos.

Abordagem	Número de máquinas	Número de processos	Disposição das sequências
i2 e **i1**	4	12	Na máquina 1, 3 processos são alocados (MAR e 2 Nós AGB). Na máquina 2, mais 3 processos são alocados (2 Nós AGB, MBR). Na máquina 3, são alocados 2 processos (2 Nós AGB). Na máquina 4, são alocados 4 processos (MBR e 4 Nó AGB).
p9	4	10	Na máquina 1, 4 processos são alocados (Coord e 3 Nó AGB). Nas outras máquinas, são alocados 2 processos Nós AGB para cada.
p3	2	4	À cada máquina, são alocados dois processos.
seq	1	1	O processo é alocado a uma única máquina.

Tabela 6.1: *Disposição dos processos das estratégias i2, i1, p9, p3 e seq nos processadores do Cluster-MP e do Cluster-IE.*

A distribuição dos processos ficou a cargo do MPICH, que tenta distribuir os processos de forma balanceada.

As estratégias i2 e i1 implementadas possuem 12 processos, onde o MAR possui identificador igual a 0, os 3 Nós AGB da SIAR possuem identificadores iguais a 1, 2 e 3, o MBR da SIBR 1 possui identificador igual a 4, os 3 Nós AGB nessa Super Ilha possuem identificadores iguais a 5, 6 e 7, o MBR da SIBR 2 possui identificador igual a 8 e os 3 Nós AGB possuem identificadores iguais a 9, 10 e 11. A estratégia p9 possui 10 processos, onde o processo Coord (Figura 6.1.c) possui identificador igual a 0 e os outros 9 processos Nós AGB recebem identificadores de 1 a 9. Da mesma forma, são distribuídos os identificadores de p3, p3 tem 4 processos, onde Coord tem identificador igual a 0 e os outros 3 Nós AGB recebem identificadores de 1 a 3. A estratégia seq possui 1 processo com identificador igual a 0.

Foram utilizados conjuntos de sequências reais para os testes, apresentadas na Tabela 6.2. Cada conjunto foi escolhido tendo por base suas características, de modo que os algoritmos pudessem ser avaliados em diferentes aspectos, como tendência à convergir para ótimos locais (Capítulo 2), velocidade de melhoria do alinhamento, entre outros. A base real *1ac5* (http://bips.u-strasbg.fr/fr/Products/Databases/BAliBASE/) foi selecionada pelo seu alto grau de complexidade, pois o conjunto de suas sequências confunde algoritmos heurísticos, direcionando-os para ótimos locais, possibilitando verificar com mais facilidade as estratégias com propensão à convergência prematura. A base real *ttkrsyedq* (http://bips.u-strasbg.fr/fr/Products/Databases/BAliBASE/) é composta de sequências similares de fácil alinhamento, dessa forma o algoritmo não se confunde, ficando evidenciado quais algoritmos convergem mais rapidamente para um resultado

Base	Nome	Número de sequências	Tamanho médio das sequências
BAliBASE [54, 1]	*1ac5*	4	445
PFAM [6]	*ttkrsyedq*	3	513
PFAM [6]	*virul fac*	15	820

Tabela 6.2: *Sequências utilizadas para realização dos testes.*

próximo do esperado. A base real *virul fac* (http://pfam.sanger.ac.uk/) é uma base composta por um maior número de sequências de maior comprimento, dessa forma tentaremos determinar como as estratégias se comportam para sequências grandes.

6.2 Resultados Obtidos

Nessa seção, serão apresentados os resultados experimentais obtidos com as bases reais *1ac5*, *ttkrsyedq* e *virul fac*. Devido às diferenças entre as estratégias utilizadas (Figura 6.1), tornou-se complexo determinar alguns parâmetros, tais como tamanho da população inicial e intervalo de Migração, de forma a estabelecer uma comparação objetiva entre tempo de execução e qualidade dos resultados obtidos. O tamanho da população foi definido empiricamente, de maneira que a população não fosse ficar muito grande para um nó processar (seq) nem muito pequena quando fosse dividida em vários nós (i2, i1, p9 e p3). Para determinar o tamanho da população, foi considerado qual a capacidade de processamento e memória cada nó possui, o tempo de processamento e tempo para migração de dados. O intervalo de Migração também foi definido empiricamente, a partir de alguns testes, descritos na seção 6.2.4. Os intervalos de Migração Interna igual a 10 gerações e o Migração Externa de 50 foram considerados adequados.

Os testes apresentados nas seções 6.2.1, 6.2.2 e 6.2.3 utilizarão a seguinte nomenclatura: $< nome\_base > - < maquina > - < numero\_de\_geracoes >$. Por exemplo, o teste da base real *1ac5* no Cluster-MP com 5000 gerações será nomeado 1ac5-MP-5000.

6.2.1 Avaliação da base 1ac5

Teste 1ac5-MP-5000: Esse teste foi realizado na base real *1ac5* (Tabela 6.2) no Cluster-MP para 5000 gerações. As estratégias utilizadas foram: i1, p9, p3 e seq (seção 6.1). Nesse teste, a população total de cada estratégia foi calculada com base no número de Nós AGB que ela possui. Cada Nó de AGB tem a população de 20 elementos, de modo que as estratégias possuem populações totais com número de indivíduos distintos. Por exemplo, a estratégia p9 tem a população total de 180 enquanto que p3 tem a população de 60. Esses dados são apresentados na Tabela 6.3.

Conforme os resultados apresentados na Tabela 6.4 e o gráfico ilustrado pela Figura 6.2, é observado que, para todas as estratégias, nas primeiras 500 gerações

Abordagem	**Processos AGB**	**Processos (MAR, MBR ou Coord)**	**População Total**	**População por Filho**	**Aptidão Inicial**	**Aptidão Final**
i1	9	3	180	20	$-369,73$	**-202,75**
p9	9	1	180	20	$-377,73$	$-230,1$
p3	3	1	60	20	$-387,67$	$-273,6$
seq	1	0	20	20	$-429,5$	*-312,88*

Tabela 6.3: *Tabela final de resultados dos testes realizados na base real 1ac5 no Cluster-MP para* 5000 *gerações. Em negrito, consta o melhor resultado de aptidão obtido e em itálico, o pior.*

houve maior parte do aumento do valor de aptidão, sendo que a estratégia i1 apresentou aumento mais acelerado. No intervalo de 500 à 1000 gerações, o aumento da aptidão em p3 e p9 são maiores que o aumento em i1, mostrando que, apesar do crescimento de i1 ter sido mais acelerado, p3 e p9 estão crescendo continuamente, mesmo que de forma mais lenta. Os resultados obtidos neste experimento para 5000 gerações mostraram um crescimento contínuo de cada estratégia. Além disso, nenhuma das estratégias convergiu prematuramente para um valor, continuando a gerar soluções diferentes e melhores no decorrer das gerações.

Conforme o esperado, o algoritmo seq obteve o resultado final pior do que a estratégia p3, que, por sua vez, obteve o resultado pior do que a estratégia p9, por se tratarem de estratégias com populações totais menores e com um número de Nós AGB menor, ou seja, menor quantidade de processamento. O melhor resultado obtido foi pela estratégia i1 ($-202,75$) que foi $11,89\%$ melhor que p9 ($-230,1$), $25,9\%$ melhor que p3 ($-273,6$) e $35,20\%$ melhor que seq ($-312,88$). A estratégia p9 obteve o resultado pior que o de i1, apesar do algoritmo de ambos apresentarem populações de um mesmo tamanho, um mesmo número de Nós AGB e p9 ser composto somente de nós de alta resolução (que evoluem a sequência inteira). Isso mostra que, nesse contexto, a estratégia de Injeção de Ilhas trouxe melhorias em relação à estratégia de ilhas convencional.

Número de gerações	**i1**	**p9**	**p3**	**seq**
0	$-369,73$	$-377,73$	$-387,67$	$-429,5$
500	$-278,79$	$-333,33$	$-329,08$	$-374,08$
1000	$-266,6$	$-281,02$	$-314,21$	$-352,77$
1500	$-261,65$	$-266,75$	$-302,73$	$-347,25$
2000	$-246,4$	$-261,71$	$-300,44$	$-341,42$
2500	$-238,33$	$-255,96$	$-297,54$	$-335,02$
3000	$-225,29$	$-251,73$	$-288,17$	$-330,42$
3500	$-213,94$	$-248,06$	$-276,94$	$-328,19$
4000	$-206,81$	$-244,38$	$-274,81$	$-321,54$
4500	$-205,56$	$-237,17$	$-274,25$	$-317,44$
5000	**-202,75**	$-230,1$	$-273,6$	*-312,88*

Tabela 6.4: *Resultados na base real 1ac5 no Cluster-MP para* 5000 *gerações. Em negrito, consta o melhor resultado obtido e em itálico, o pior.*

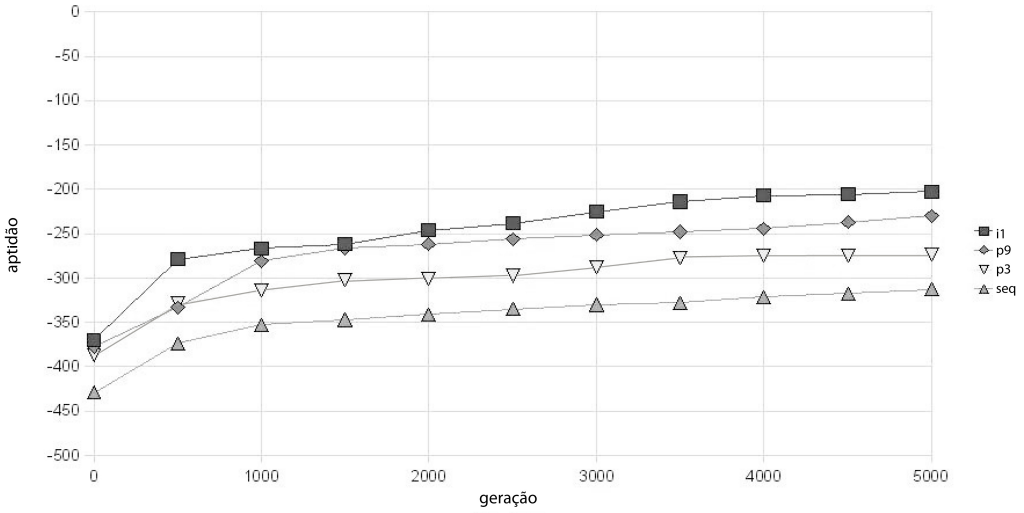

Figura 6.2: *Gráfico dos resultados do teste 1ac5-MP-5000 presentes na Tabela 6.4.*

Teste 1ac5-IE-5000: Esse teste foi realizado na base real *1ac5* (Tabela 6.2) no Cluster-IE para 5000 gerações. As estratégias utilizadas foram: i2, i1, p9, p3 e seq (seção 6.1). As populações totais de cada estratégia são iguais nesse teste, então os Nós AGB de estratégias distintas podem ter populações de tamanhos diferentes. As estratégias, i2, i1 e p9 possuem a mesma população para cada Nó AGB. Esses dados são apresentados na Tabela 6.5.

Conforme os resultados apresentados na Tabela 6.6 e o gráfico ilustrado pela Figura 6.3, constata-se que nas primeiras 500 gerações o crescimento de todas as estratégias é bastante acelerado, sendo p9 a estratégia que obteve maior aumento,

seguida de i1, i2, p3 e seq. A estratégia p9 aumentou quase duas vezes (1, 97 vezes) o valor que a estratégia seq aumentou. Apesar de p9 ter obtido o maior aumento, i1 ficou com um valor maior que p9, tendo, também, um aumento bem próximo ao dele, podendo ser o aumento elevado de p9 devido a ele ter partido de um valor muito baixo. Nas próximas 1000 gerações, as estratégias continuam a evoluir em um ritmo parecido entre si e menos acelerado, com exceção de seq, que parte de uma evolução lenta e pula para um resultado alto ficando, momentaneamente, com um valor maior que quase todas as estratégias (menos que i1). Geralmente isso ocorre quando um algoritmo está com um população viciada e um operador de mutação consegue gerar uma solução nova e boa. Entre gerações 2500 e 4000, a maioria das estratégias se estabiliza não aumentando o valor, ou aumentando de forma muito baixa. No entanto, i2 fica durante um intervalo curto estável, e retoma um crescimento relativamente acelerado. Nas últimas 1000 gerações, i2 cresce bastante, i1 e seq crescem pouco.

Como resultado final, i2 ($-149,06$) obteve um valor bem superior quando comparado ao das outras estratégias, $23,48\%$ maior que i1 ($-194,81$), $24,96\%$ maior que seq ($-198,63$), $29,48\%$ maior que p3 ($-211,81$) e $31,11\%$ maior que p9 ($-216,38$). A estratégia i1 ($-194,81$) obteve o segundo melhor resultado mais próximo das outras estratégias, $1,92\%$ maior que seq ($-198,63$), $8,03\%$ maior que p3 ($-211,81$) e $9,97\%$ maior que p9 ($-216,38$). Como já mencionado anteriormente no teste 1ac5-MP-5000, o fato de p9 ter obtido novamente resultados piores que i1 e também piores que i2, reforça a idéia de que a estratégia Injeção de Ilhas trouxe melhorias em relação à estratégia de ilhas convencional.

As estratégias i1 e p9 seguem um comportamento semelhante nesse teste e no teste 1ac5-MP-5000, considerando uma estratégia em relação à outra. Nas gerações iniciais ambas as estratégias possuem uma evolução acelerada, depois ambas continuam evoluindo de forma menos acelerada, sendo que i1 sempre está acima de p9.

Na Tabela 6.5, encontra-se o tempo de execução de cada estratégia. As estratégias i2 e i1 possuem tempos próximos de 3127 e 3232 segundos respectivamente. Em relação à p9, i2 e i1 são em média 5 vezes ($5,16$ e $4,99$ vezes) mais rápidos. Um dos motivos que poderia ser a causa do tempo elevado de p9 é o fato de p9 ser composto somente de nós de alta resolução (que evoluem a sequência inteira), enquanto i2 e i1 processam somente parte das sequências nas SIBR. No entanto, esse fato aparentemente não justificaria, pois apesar de i2 e i1 processarem somente parte das sequências nas SIBR, as populações de partes são maiores (maior número de indivíduos) que as de sequências inteiras acarretando nisso um gasto de tempo. Também, existem dois processos a mais em i2 e i1 que executam etapas de corte e junção de sequências também consumindo tempo. Outro motivo que poderia ser a causa do tempo elevado de p9 é a estratégia de migração adotada, pois, no intervalo de Migração, o algoritmo do Nó AGB é bloqueado até que ele receba os indivíduos do vizinho da esquerda. Essa espera pode acarretar em um gasto elevado de tempo. Quando os tempos de i2 e i1 são comparados com o de p3, é observado que p3 é aproximadamente 10 vezes ($10,37$ e $10,03$) mais lento que i2 e i1. Parte do tempo de execução elevado de p3 se deve ao fato de p3 possuir um número de Nós AGB 3 vezes menor que i2 e i1 para processar o mesmo tamanho de população. A estratégia seq é, aproximadamente, 2 vezes

Abordagem	**Processos AGB**	**Processos (MAR, MBR ou Coord)**	**População Total**	**População por Filho**	**Tempo (s)**	**Aptidão Inicial**	**Aptidão Final**
i2	9	3	180	20	3127	$-369,5$	**-149,06**
i1	9	3	180	20	3232	$-360,81$	$-194,81$
p9	9	1	180	20	16142	$-382,63$	*-216,38*
p3	3	1	180	60	32411	-355	$-211,81$
seq	1	0	180	180	6774	$-349,06$	$-198,63$

Tabela 6.5: *Tabela final de resultados dos testes realizados na base real 1ac5 no Cluster-IE para* 5000 *gerações. Em negrito, consta o melhor resultado de aptidão obtido e em itálico, o pior.*

Número de gerações	**i2**	**i1**	**p9**	**p3**	**seq**
0	$-369,5$	$-360,81$	$-382,63$	-355	$-349,06$
500	-281	$-257,81$	$-272,25$	$-282,56$	$-293,13$
1000	$-259,69$	$-239,13$	$-255,81$	$-264,13$	$-286,19$
1500	$-246,25$	-226	$-239,06$	$-249,06$	$-234,63$
2000	$-227,88$	$-225,56$	$-233,31$	$-234,31$	$-219,69$
2500	$-227,06$	$-219,94$	$-232,13$	$-221,75$	$-219,69$
3000	$-227,06$	$-218,38$	$-232,13$	$-216,69$	$-218,38$
3500	-219	$-217,31$	$-229,81$	$-215,81$	$-214,06$
4000	$-189,06$	$-216,88$	$-229,69$	$-213,81$	-212
4500	$-153,88$	$-208,25$	$-219,44$	$-213,81$	$-208,94$
5000	**-149,06**	$-194,81$	*-216,38*	$-211,81$	$-198,63$

Tabela 6.6: *Resultados na base real 1ac5 no Cluster-IE para* 5000 *gerações. Em negrito, consta o melhor resultado obtido e em itálico, o pior.*

($2,17$ e $2,1$) mais lenta que i2 e i1. Ela trabalha com uma população de 180 indivíduos em um único processador, enquanto i2 e i1 dividem essa população em 9 Nós AGB que estão distribuídos entre 8 processadores. Cada Nó AGB executa o mesmo número de gerações que o seq. Então, apesar de em i1 e i2, os Nós AGB estarem executando o mesmo número de gerações que seq em menos de um processador por processo, quando comparado à seq, as estratégias i1 e i2, executadas em 8 processadores, ainda são mais rápidas, pois lidam com um população menor em cada Nó AGB.

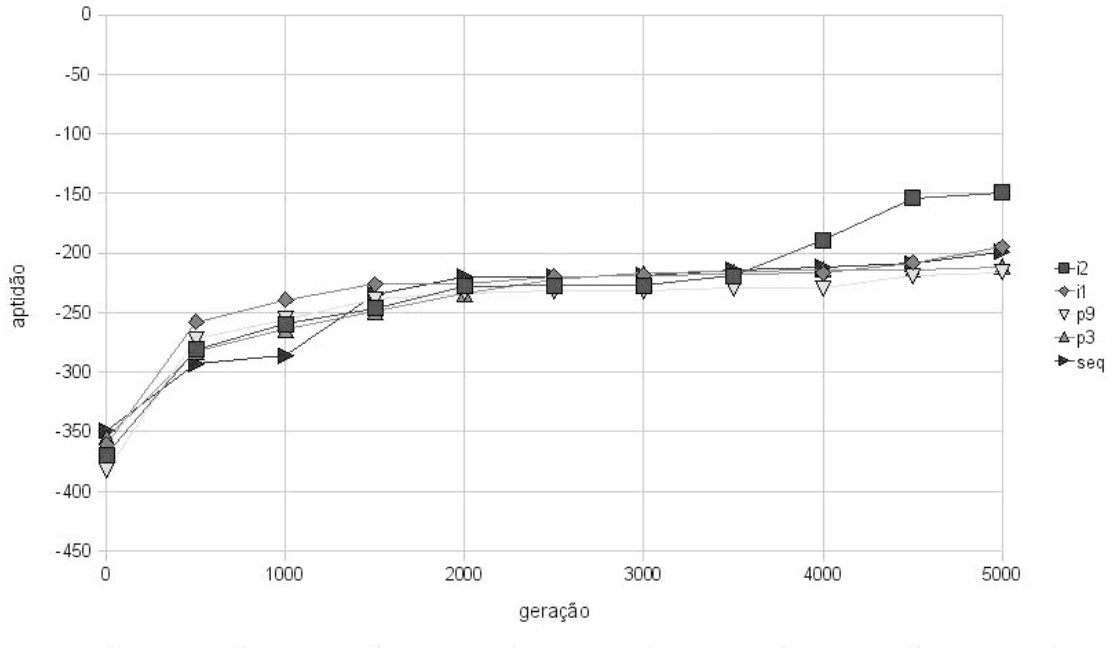

Figura 6.3: *Gráfico dos resultados do teste 1ac5-IE-5000 presentes na Tabela 6.6.*

Teste 1ac5-IE-10000: Esse teste foi realizado na base real *1ac5* (Tabela 6.2) no Cluster-IE para 10000 gerações. As estratégias utilizadas foram: i2, i1, p9, p3 e seq (seção 6.1). As populações totais de cada estratégia são iguais. Esses dados são apresentados na Tabela 6.7.

Conforme os resultados apresentados na Tabela 6.8 e o gráfico ilustrado pela Figura 6.4, é observado que nas primeiras 1000 gerações todas as estratégias apresentam um crescimento elevado e próximo, com exceção da seq que apresenta um crescimento, em média, 48% abaixo do crescimento das outras estratégias. Ao final de 1000 gerações, a estratégia que possui maior valor é a i2 ($-193,31$) seguida das p3 ($-203,56$), i1 (-218), seq (-266) e p9 ($-270,44$). No intervalo de 1000 à 5000, as estratégias i2, i1 e p3 continuam crescendo em um ritmo constante e p9 e seq ficam estáveis, crescendo pouco, de forma que cada estratégia mantém sua posição em relação a outra. Assim como no teste 1ac5-IE-5000, i2 ($-141,63$) também se mantém na melhor posição na geração 5000. No entanto, dessa vez a estratégia não está com um valor tão superior às estratégias i1 ($-165,44$) e p3 ($-154,31$), sendo i2 $14,39\%$ e $8,22\%$ maior que i1 e p3 respectivamente. No entanto, i2 continua com uma grande vantagem em relação à p9 ($-239,38$) e seq ($-225,81$), de $40,84\%$ e $27,28\%$ respecivamente.

Nas próximas 5000 gerações, p3 cresce mais rapidamente e supera i2 assumindo o maior valor de $-107,88$. A estratégia i1 também cresce em um ritmo mais rápido e apesar de em todo o intervalo de 10000 gerações está abaixo de i2, no final, ela fica bem próxima de i2, com o valor final de i1 igual a $-126,38$ e o de i2 igual a $-127,94$ (diferença de $1,04\%$). As estratégia p9 apresenta um crescimento constante enquanto seq permanece estável, até que, na geração 7000, p9 cresce de forma mais acelerada, superando o seq e, em seguida, ambos ficam

Abordagem	**Processos AGB**	**Processos (MAR, MBR ou Coord)**	**População Total**	**População por Filho**	**Tempo (s)**	**Aptidão Inicial**	**Aptidão Final**
i2	9	3	180	20	6990	$-351,81$	$-127,94$
i1	9	3	180	20	7912	$-361,88$	$-126,38$
p9	9	1	180	20	33430	$-412,81$	-205
p3	3	1	180	60	64444	$-356,94$	**-107,88**
seq	1	0	180	180	13903	$-338,31$	*-212,69*

Tabela 6.7: *Tabela final de resultados dos testes realizados na base real 1ac5 no Cluster-IE para* 10000 *gerações. Em negrito, consta o melhor resultado de aptidão obtido e em itálico, o pior.*

estáveis, com crescimento baixo. Esses pulos no gráfico, como o do crescimento do p9, geralmente acontece quando a estratégia encontra uma solução diferente por meio da aplicação dos operadores de mutação e essa solução é bem melhor que as anteriores. Ao final, p3 ($-107,88$) obtém o melhor resultado, sendo $14,64\%$ superior a i1, $15,68\%$ superior a i2, $47,38\%$ superior a p9 e $49,28\%$ superior a seq.

A estratégia p3 mostra uma evolução rápida e um resultado superior a todas as outras estratégias e bem superior à p9 ($47,38\%$ superior), apesar de possuírem a mesma população total e p9 evoluir a população em um número maior de nós de Nós AGB. Pode-se concluir, que o tamanho da população de cada nó é um fator que influencia na qualidade do resultado população, sendo, que às vezes, é mais interessante ter um número menor de nós com uma população maior. Nos resultados obtidos por seq, observa-se uma relação contrária à anterior, onde existe um nó com uma população grande convergindo prematuramente. Logo, a relação tamanho da população e número de nós deve ser equilibrada de forma a obter resultados melhores.

Conforme mencionado anteriormente nos testes 1ac5-MP-5000 e 1ac5-IE-5000, o fato de p9 ter obtidos novamente resultados piores que i1 e também piores que i2, reforça a idéia que a estratégia Injeção de Ilhas trouxe melhorias em relação à estratégia de ilhas convencional.

Assim como no teste 1ac5-IE-5000, também nesse teste, as estratégias i1 e p9 seguem um comportamento parecido nesse teste e no teste 1ac5-MP-5000, pois nas gerações iniciais ambas as estratégias possuem uma evolução acelerada, depois ambas continuam evoluindo de forma menos acelerada, sendo que i1 sempre está acima de p9.

Os tempos execução de cada estratégia, apresentados na Tabela 6.5, apresentam comportamentos aproximados aos do teste 1ac5-IE-5000. Novamente i2 e i1 obtêm tempos próximos de 6990 e 7912 segundos respetivamente. As estratégias i2 e i1 são $4,78$ e $4,22$ vezes mais rápidas que p9, $9,22$ e $8,14$ mais rápidos que p3 e $1,99$ e $1,76$ mais rápidos que seq, respectivamente.

Número de Gerações	**i2**	**i1**	**p9**	**p3**	**seq**
0	$-351,81$	$-361,88$	$-412,81$	$-356,94$	$-338,31$
500	$-266,75$	$-241,63$	$-331,5$	-242	$-279,56$
1000	$-193,31$	-218	$-270,44$	$-203,56$	-266
1500	$-181,5$	$-210,44$	$-264,06$	$-178,06$	$-245,88$
2000	$-174,63$	-203	$-262,88$	$-174,19$	$-236,44$
2500	$-167,75$	$-196,94$	$-250,94$	$-167,31$	$-231,31$
3000	$-162,31$	$-188,5$	$-249,19$	$-165,94$	$-231,31$
3500	$-159,69$	$-179,31$	$-246,88$	$-165,81$	$-228,25$
4000	$-147,81$	$-170,5$	$-246,75$	$-160,75$	$-225,81$
4500	$-143,25$	$-167,5$	$-241,56$	$-157,13$	$-225,81$
5000	$-141,63$	$-165,44$	$-239,38$	$-154,31$	$-225,81$
5500	$-140,75$	-148	$-238,19$	$-145,31$	$-223,31$
6000	$-140,63$	$-147,38$	$-229,75$	$-141,81$	$-216,56$
6500	$-134,13$	$-145,31$	$-227,25$	$-132,13$	$-215,56$
7000	$-134,06$	$-140,56$	$-209,44$	$-126,06$	$-215,38$
7500	$-134,06$	$-140,31$	$-208,63$	$-124,56$	$-215,06$
8000	$-133,75$	$-140,31$	$-207,38$	$-121,81$	$-212,69$
8500	$-133,75$	$-139,69$	$-205,31$	$-114,38$	$-212,69$
9000	$-133,75$	$-139,25$	$-205,31$	$-108,94$	$-212,69$
9500	$-131,5$	$-135,75$	$-205,13$	$-108,06$	$-212,69$
10000	$-127,94$	$-126,38$	-205	**-107,88**	*-212,69*

Tabela 6.8: *Resultados na base real 1ac5 no Cluster-IE para* 10000 *gerações. Em negrito, consta o melhor resultado obtido e em itálico, o pior.*

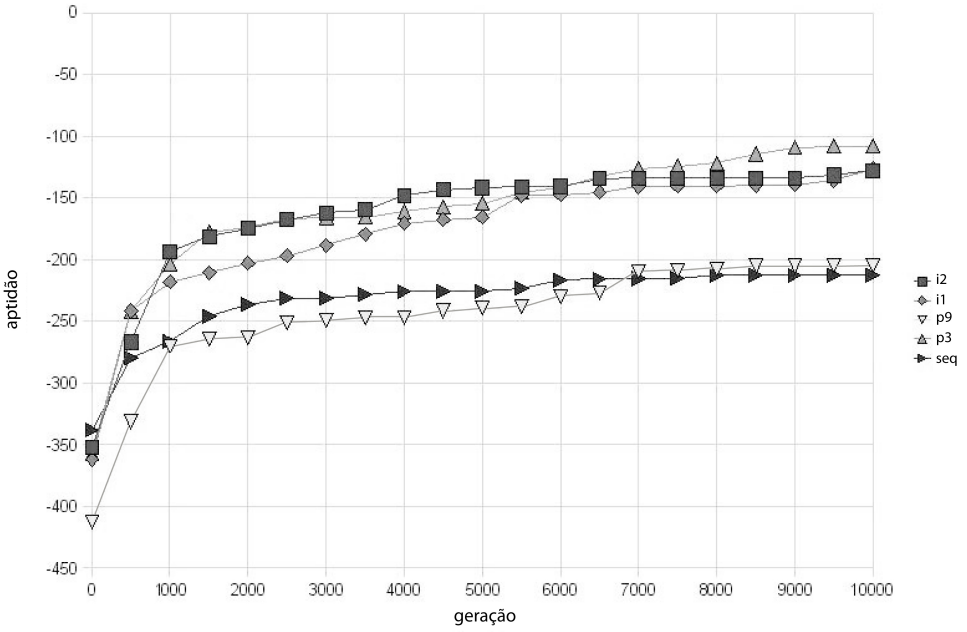

Figura 6.4: *Gráfico dos resultados do teste 1ac5-IE-10000 presentes na Tabela 6.8.*

6.2.2 Avaliação da base ttkrsyedq

Teste ttkrsyedq-MP-5000: Esse teste foi realizado na base real *ttkrsyedq* (Tabela 6.2) no Cluster-MP para 5000 gerações. As estratégias utilizadas foram: i1, p9, p3 e seq (seção 6.1). Nesse teste, a população total de cada estratégia foi calculada com base no número de Nós AGB que ela possui. Esses dados são apresentados na Tabela 6.3.

Conforme mencionado anteriormente, a base real *ttkrsyedq* é composta de sequências similares fáceis de alinhar, dessa forma possibilitando identificar qual estratégia converge mais rápido para um resultado próximo do esperado. Pelos resultados apresentados na Tabela 6.10 e pelo gráfico ilustrado pela Figura 6.5, observa-se que após 2000 gerações as estratégias i1 $(587,59)$, p9 $(586,3)$ e p3 $(585,44)$ já convergiram para o resultado. A estratégia i1 $(581,63)$, já quase converge nas primeiras 500 gerações, tendo ela já convergido na geração 1000, enquanto p9 e p3 só convergem na geração 2000. A estratégia seq não converge no intervalo de 5000 gerações. Nesse teste, a estratégia i1 convergiu em menos gerações para um resultado bom, seguida de p3 e p9. No final, a estratégia p3 $(589,56)$ obtém o melhor resultado, seguida de p9 $(589,33)$, i1 $(587,81)$ e seq $(555,67)$. Com a diferença entre os resultados de i1, p9 e p3 é muito pequena $(0,3\%$ e $0,26\%$ respectivamente), podemos dizer que as três estratégias conseguiram convergir para resultados muito próximos.

Nesse teste, a estratégia de Injeção de Ilhas i1 trouxe melhorias em relação às estratégias de ilha convencionais p9, p3 e seq, convergindo, em um número menor de gerações, para o valor esperado.

Abordagem	**Processos AGB**	**Processos (MAR, MBR ou Coord)**	**População Total**	**População por Filho**	**Aptidão Inicial**	**Aptidão Final**
i1	9	3	180	20	243, 04	587, 81
p9	9	1	180	20	259, 19	589, 33
p3	3	1	60	20	262, 07	**589,56**
seq	1	0	20	20	226, 3	*555,67*

Tabela 6.9: *Tabela final de resultados dos testes realizados na base real ttkrsyedq no Cluster-MP para* 5000 *gerações. Em negrito, consta o melhor resultado de aptidão obtido e em itálico, o pior.*

Número de gerações	**i1**	**p9**	**p3**	**seq**
0	243, 04	259, 19	262, 07	226, 3
500	581, 63	520, 63	458, 44	370, 11
1000	587, 37	562, 15	521, 44	396, 07
1500	587, 59	583, 52	570, 85	436, 78
2000	587, 59	586, 3	585, 44	446, 22
2500	587, 59	587, 33	586, 04	483, 89
3000	587, 59	588	587, 07	512, 41
3500	587, 59	588	588, 15	517, 89
4000	587, 59	588, 89	588, 74	541, 63
4500	587, 59	589, 19	589, 56	553, 81
5000	587, 81	589, 33	**589,56**	*555,67*

Tabela 6.10: *Resultados na base real ttkrsyedq no Cluster-MP para* 5000 *gerações. Em negrito, consta o melhor resultado obtido e em itálico, o pior.*

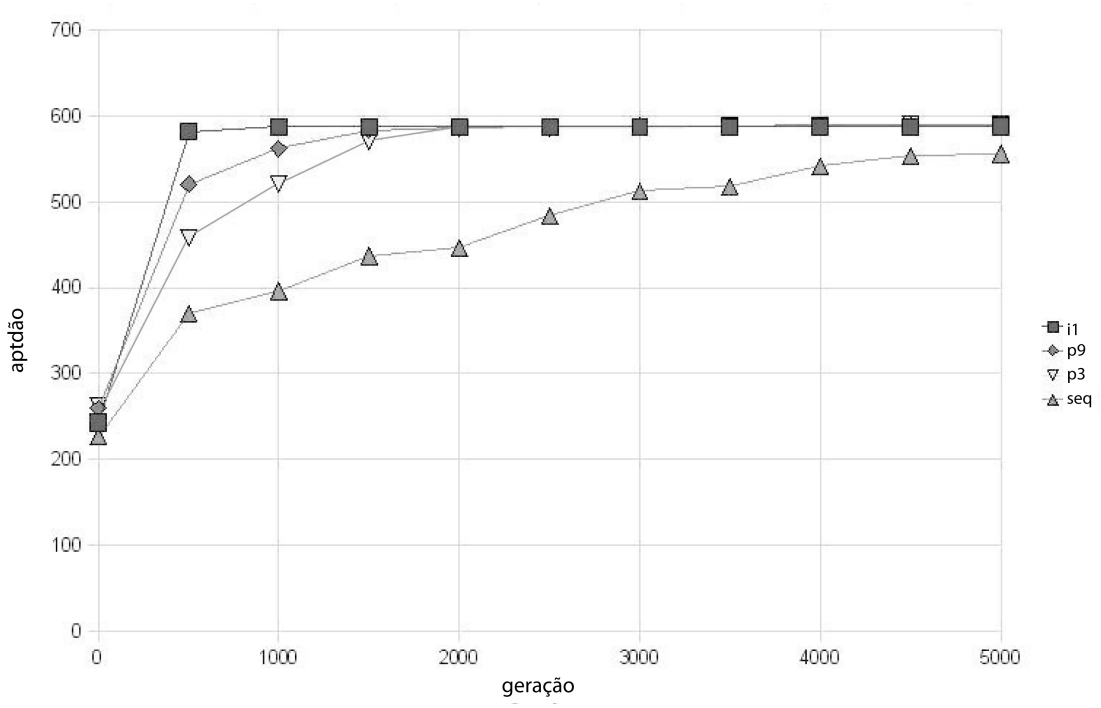

Figura 6.5: *Gráfico dos resultados do teste ttkrsyedq-MP-5000 presentes na Tabela 6.10.*

6.2.3 Avaliação da base virul fac

Teste virul_fac-IE-5000: Esse teste foi realizado na base real *virul fac* (Tabela 6.2) no Cluster-IE para 5000 gerações. As estratégias utilizadas foram: i2, i1 e p9 (seção 6.1). As populações totais de cada estratégia são iguais. Esses dados são apresentados na Tabela 6.11.

Conforme apresentado na Tabela 6.2, a base real *virul fac* é bem maior que as outras, tanto em número como no comprimento das sequências. Essas sequências, no entanto, não confundem o algoritmo. Pode-se observar nos resultados apresentados na Tabela 6.12 e no gráfico ilustrado pela Figura 6.6, que, em todo o intervalo, i2 apresenta resultados melhores que p9 que, por sua vez, apresenta resultados melhores que i1. Ao final de 5000 gerações, i2 $(-530,74)$ obtém o melhor resultado, sendo $7,22\%$ melhor que p9 $(-571,44)$ e $8,65\%$ melhor que i1 $(-580,98)$. As estratégias p9 e i1 apresentam resultados finais aproximados. Assim como nos outros testes (1ac5-MP-5000, 1ac5-IE-5000, 1ac5-IE-10000 e ttkrsyedq-MP-5000), as estratégias i2, i1 e p9 apresentam um crescimento elevado nas primeiras 500 gerações.

Diferente dos testes 1ac5-IE-5000 e 1ac5-IE-10000, os tempos de execução das estratégias i2 (67273 segundos) e i1 (90513 segundos) foram maiores que os da estratégia p9 (53000 segundos), $1,27$ e $1,71$ vezes maiores respectivamente. Isso se deve principalmente ao aumento do tamanho da sequências, que implica em um gasto de tempo maior nas migrações, especialmente dentro das SIBR que são

Abordagem	**Processos AGB**	**Processos (MAR, MBR ou Coord)**	**População Total**	**População por Filho**	**Tempo (s)**	**Aptidão Inicial**	**Aptidão Final**
i2	9	3	180	20	67273	$-369,5$	**-530,74**
i1	9	3	180	20	90513	$-360,81$	*-580,98*
p9	9	1	180	20	53000	$-382,63$	$-571,44$

Tabela 6.11: *Tabela final de resultados dos testes realizados na base real virul fac no Cluster-IE para* 5000 *gerações. Em negrito, consta o melhor resultado de aptidão obtido e em itálico, o pior.*

Número de gerações	**i2**	**i1**	**p9**
0	$-711,5$	$-747,3$	$-740,98$
500	$-591,2$	$-669,06$	$-655,35$
1000	$-571,31$	$-659,23$	$-611,93$
1500	$-565,9$	$-637,84$	$-597,47$
2000	$-554,39$	$-633,76$	$-589,13$
2500	$-551,51$	$-632,82$	$-584,84$
3000	$-547,61$	$-598,52$	$-580,68$
3500	$-540,38$	$-596,49$	$-577,33$
4000	$-538,62$	$-594,36$	$-573,96$
4500	$-531,97$	$-593,03$	$-571,97$
5000	**-530,74**	*-580,98*	$-571,44$

Tabela 6.12: *Resultados na base real virul fac no Cluster-IE para* 5000 *gerações. Em negrito, consta o melhor resultado obtido e em itálico, o pior.*

migrados blocos maiores de dados nos intervalos de Migração Interna e Externa.

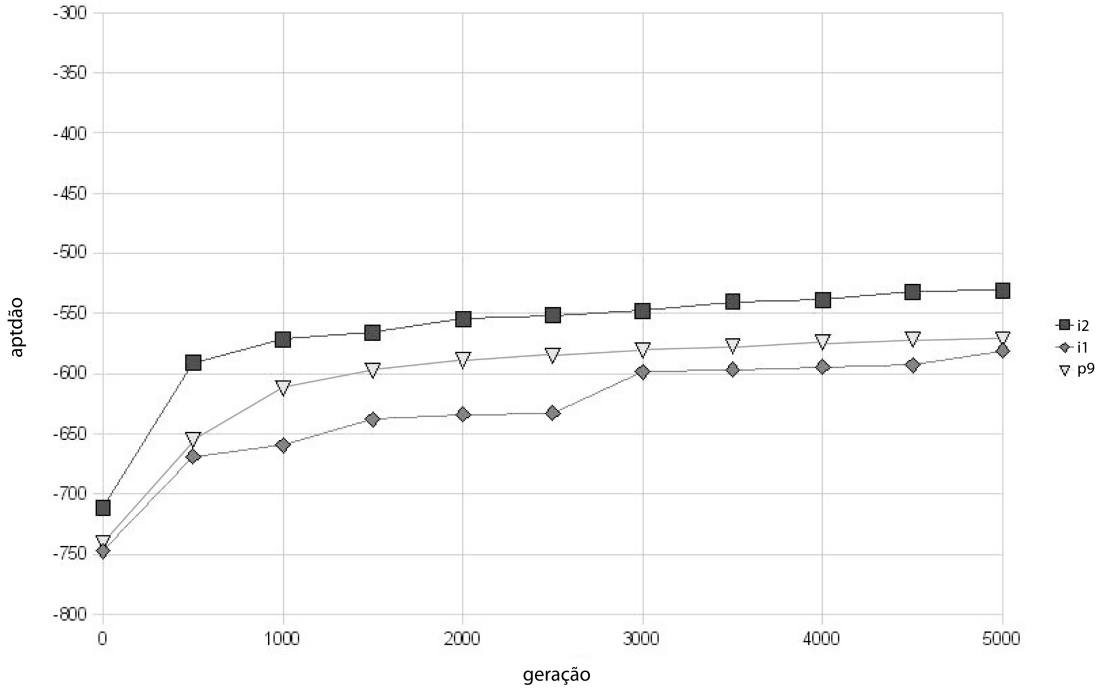

Figura 6.6: *Gráfico dos resultados do teste virul_fac-IE-5000 presentes na Tabela 6.12.*

6.2.4 Avaliação do Intervalo de Migração

Nessa seção, será avaliada a relação da variação dos intervalos de Migração Interna (IMI) e Migração Externa (IME) com o valor de aptidão final obtido, para as estratégias i2 e i1, nas bases reais *1ac5* e *virul fac*.

A Tabela 6.13 mostra os valores de aptidão obtidos com diversos intervalos de migração, estratégias e bases biológicas. Para a execução da abordagem i2 com a base real *1ac5*, os melhores intervalos de Migração Interna e Externa foram 20 e 100, respectivamente. Na execução da abordagem i1 sobre a mesma base, os melhores intervalos foram 20 (IMI) e 200 (IME). Já na execução da abordagem i1 sobre a base *virul fac*, os melhores resultados foram obtidos para 10 (IMI) e 100 (IME). Os piores resultados foram obtidos para 20 (IMI) e 200 (IME), i2-1ac5; 10 (IMI) e 200 (IME), i1-1ac5; e 20 (IMI) e 100 (IME), i1-virul_fac. Os operadores 10 (IMI) e 50 (IME) sempre levaram a valores finais intermediários.

Portanto, pode-se notar que: a) não existe uma relação dos valores dos intervalos de migração que obtem resultados bons ou ruins com as estratégias nem com as bases; b) uma fase de "*tuning*" é necessária para que os parâmetros mais adequados sejam escolhidos, pois uma pequena variação nos mesmos tem grande influência nos resultados obtidos; c) a escolha dos valores 10 (IMI) e 50 (IME) para os testes das seções 6.2.1, 6.2.2 e 6.2.3, foi acertada pois, para as bases avaliadas, esses parâmetros levaram a resultados finais intermediários (nem os melhores, nem os piores), o que mostra que seus valores são apropriados como primeira escolha.

Abordagem	Base	IMI	IME	Aptidão
i2	*1ac5*	10	50	$-149,06$
i2	*1ac5*	20	100	**-98,00**
i2	*1ac5*	10	100	$-232,19$
i2	*1ac5*	10	200	$-177,75$
i2	*1ac5*	20	200	*-181,75*
i1	*1ac5*	10	50	$-194,81$
i1	*1ac5*	20	100	$-188,00$
i1	*1ac5*	10	100	$-209,13$
i1	*1ac5*	10	200	*-222,63*
i1	*1ac5*	20	200	**-138,19**
i1	*virul fac*	10	50	$-580,98$
i1	*virul fac*	20	100	*-610,05*
i1	*virul fac*	10	100	**-538,98**
i1	*virul fac*	10	200	$-580,83$
i1	*virul fac*	20	200	$-595,78$

Tabela 6.13: *Tabela comparativa do comportamento das estratégias i2 e i1 em diferentes intevalos de migração. Em negrito, consta o melhor resultado de aptidão obtido e em itálico, o pior.*

6.2.5 Discussão

Nesse capítulo, foram realizados testes para validar as estratégias de Injeção de Ilhas (i2 e i1) propostas, em relação às estratégias de Ilhas (p9 e p3) e à estratégia Sequencial (seq).

Foram realizados testes em três bases reais (Tabela 6.2) com características específicas, a fim de testar diferentes aspectos das estratégias.

A base real *1ac5* possui sequências que são difíceis de serem alinhadas, pois induzem os algoritmos para ótimos locais. Por ter essa característica, a base pode direcionar a estratégia por diferentes caminhos podendo demorar mais ou menos para se chegar a um resultado, dificultando que seja estabelecido um padrão de comportamento para uma dada estratégia na mesma. Essa falta de padrão de comportamento pode ser observada nos testes 1ac5-IE-5000 e 1ac5-IE-10000. Os testes são executados dentro das mesma condições, no entanto o número de gerações executadas no primeiro foi 5000 e no segundo é 10000. Na geração 5000, o teste 1ac5-IE-5000 tem como resultado a estratégia i2 ($-149,06$) com valor bem superior aos das outras estratégias, $23,48\%$ maior que i1 ($-194,81$), $24,96\%$ maior que seq ($-198,63$), $29,48\%$ maior que p3 ($-211,81$) e $31,11\%$ maior que p9 ($-216,38$) e as estratégias i1, p9, p3 e seq, com valores mais próximos entre si (maior diferença entre elas é de $9,97\%$). O teste 1ac5-IE-10000 tem como resultados, na geração 5000, i2 ($-141,63$), p3 ($-154,31$) e i1 ($-165,44$) com valores altos relativamente próximos (maior diferença entre eles é de $14,39\%$) e bem abaixo seq ($-225,81$) e p9 ($-239,38$), sendo que entre as sequências mais distantes (i2 e p9) a diferença chega à $40,84\%$. Nota-se que, apesar de no teste 1ac5-IE-5000, i2 ter apresentado um resultado com diferença maior em relação

as outras estratégias que em 1ac5-IE-10000, o valor final de i2 é maior no teste 1ac5-IE-10000.

Apesar dos comportamentos diversos, pode-se observar nos três testes feitos na base real *1ac5* (1ac5-MP-5000, 1ac5-IE-5000 e 1ac5-IE-10000, descritos na seção 6.2.1), que i1 e i2 (presente em dois dos três testes) sempre apresentaram um crescimento acelerado nas gerações iniciais e no final estavam entre os melhores resultados. No único teste em que os dois não obtiveram os dois melhores resultados (teste 1ac5-IE-10000), pois p3 $(-107,88)$ foi o melhor o resultado, os resultado i1 $(-126,38)$ e i2 $(-127,94)$ estavam próximos do melhor resultado, com a diferença de $14,64\%$ e $15,68\%$ respectivamente. Pela disparidade de resultados entre p3 $(-107,88)$ e p9 (-205) no teste 1ac5-IE-10000, onde p3 obteve resultados bem superiores ($47,38\%$ superior) apesar de possuir uma população total igual a p9 e menos Nós AGB, foi observado que a relação entre o tamanho da população e o número de Nós AGB que afeta os resultados. Sendo assim, deve ser feito, nesse caso, um "*tuning*" desses parâmetros (tamanho da população e número de Nós AGB), a fim de se obter resultados melhores.

A base real *ttkrsyedq* é composta por sequências de alinhamento simples, então, fica mais fácil de identificar característica inerentes da estratégia, como velocidade de convergência. No teste ttkrsyedq-MP-5000, observa-se que a estratégia i1 converge para uma solução boa em 1000 gerações, enquanto que p9 e p3 convergem em 2000 gerações e seq não converge dentro do intervalo de 5000 gerações. A estratégia i1 $(587,81)$ obtém valores finais muito próximos de p9 $(589,33)$ e p3 $(589,56)$, com a diferença maior entre esse valores de $(0,3\%)$, sendo considerado que p9, p3 e i1 convergiram para um resultado bom.

A base real *virul fac* é uma base grande com um número elevado de sequências longas. Essa base permite testar o comportamento das estratégias com um volume grande de dados. Nessa base, i2 obtém resultados melhores, seguida de p9 e i1 em todo o intervalo. Sendo os resultados finais de i1 e p9 aproximados.

Em todos os testes, com exceção do teste *virul fac*, i2 e i1 obtiveram resultados melhores que p9, apesar das populações serem do mesmo tamanho e possuírem o mesmo número de Nós AGB. Então, foi concluído que, para os testes realizados, a estratégia de Injeção de Ilhas trouxe, em média, melhorias em relação à estratégia convencional de ilhas nos testes apresentados.

A partir dos resultados bons de i2 e i1 em todos os testes, onde eles mostraram que eram capazes de crescer em direção de soluções melhores de forma rápida, sem ficar presos em ótimos locais e também com capacidade para processar soluções maiores, foi concluído que as estratégias de Injeção Ilha i1 e i2 produzem resultados em média mais aptos que as demais no contexto dos testes.

O tempo foi medido em dois testes na base real *1ac5* e no teste na base real *virul fac*. Na base real *1ac5*, i1 e i2 obtiveram tempos de execução aproximadamente 5 vezes menores que p9, 10 vezes menores que p3 e 2 vezes menores seq. A causa mais provável de tempo elevado de execução das estratégias de ilha (p9 e p3), foi o modelo de migração adotado. A pequena redução do tempo em relação à estratégia sequencial foi atribuída às diferenças estruturais entre as estratégias. No teste virul_fac-IE-5000, os tempos de execução das estratégias i2 e i1 foram maiores que os da estratégia p9, isso foi atribuído ao aumento do tamanho da sequências que, por sua vez, aumentaram o tempo gasto nas migrações.

Na seção 6.2.4, foram avaliados os intervalos de migração. A partir dos resultados, concluiu-se que não existe uma relação dos valores dos intervalos de migração que obtém resultados bons ou ruins com as estratégias nem com as bases e, como esses fatores influenciam muito no resultado, deve-se executar uma fase de "*tuning*" para que os parâmetros mais adequados sejam escolhidos. Os resultados obtidos pelo iPGA (seção 4.1) também mostram que a variação dos parâmetros de migração tem direta relação com a qualidade da população final e levantam a necessidade dos parâmetros serem testados antes de serem selecionados.

6.3 Glossário

Glossário de termos utilizados nesse capítulo:

- Injeção de Ilhas one-way (i1)
- Injeção de Ilhas two-way (i2)
- Paralela 9 ilhas (p9)
- Paralela 3 ilhas (p3)
- Sequencial (seq)

Capítulo 7

Conclusão e Trabalhos Futuros

A presente dissertação de mestrado propôs e avaliou uma estratégia Paralela Multi-ilha para o Problema do Alinhamento Múltiplo de Sequências Biológicas. A estratégia proposta foi concebida utilizando a idéia da estratégia Multi-ilha (ou de Injeção de Ilhas) apresentada em [28] e os resultados e idéias apresentados no estudo dos algoritmos genéticos paralelos para o Alinhamento Múltiplo de Sequências (Capítulo 4). Da estratégia Multi-ilha proposta em [28] foram utilizados a estrutura de três grandes ilhas, a composição de uma Ilha de Alta Resolução e outras duas de Baixa Resolução e a possibilidade das ilhas poderem trocar elementos entre si em intervalos definidos. O fato do modelo de ilha ter sido escolhido como estratégia de paralelização do AG dentro das Ilhas de Alta e Baixa Resolução, foi apoiado no estudo dos AGs paralelos (Capítulo 4), em que todas as estratégias que utilizavam o AG para resolver diretamente o problema do alinhamento eram paralelizadas pela estratégia de ilha. Partindo do pressuposto de que um indivíduo da população é formado de partes boas e ruins, surgiu a idéia de evoluir separadamente as partes dos alinhamentos, na busca de melhores soluções. Este aspecto deu origem a organização Multi-ilha proposta nesse trabalho, onde as Ilhas de Baixa Resolução evoluem separadamente partes de alinhamentos enquanto a Ilha de Alta Resolução evolui o alinhamento como um todo.

Pelo fato de, nas Ilhas de Baixa Resolução, os alinhamentos serem processados em partes, foram propostos dois operadores (seção 5.4.2), um para dividir os alinhamentos em partes, chamado operador de corte, e outro para juntar as partes formando novos alinhamentos, chamado operador de junção.

Nessa dissertação, também foram propostos dois novos operadores de mutação (seção 5.4.4): o operador *Gap Char Shifter* (GCS) e o operador *Column Gap Char Shifter* (CGCS). Ambos os operadores movem caracteres por conjuntos de espaços em branco utilizando o cálculo SP garantir que o alinhamento formado seja igual ou mais apto que o alinhamento de entrada.

Os resultados obtidos em 2 *clusters* mostaram que, em todas as bases, os algoritmos que utilizavam a estratégia de Injeção de Ilhas (i2 e i1) obtiveram resultados com aptidões superiores ou próximas das melhores, em relação às estratégias de 9 ilhas (p9), 3 ilhas (p3) e sequencial (seq). Foram avaliados critérios como capacidade de evitar ótimos locais e a velocidade de obtenção de resultados bons. Em todos eles, i2 e i1 obtiveram resultados muito bons, sendo elas

consideradas as estratégias que obtiveram resultados mais aptos como um todo. Em todos os testes, com exceção do teste na base *virul fac*, i2 e i1 obtiveram resultados muito superiores a p9, mesmo possuindo populações e um número de Nós AGB iguais.

Nos testes com a base real *1ac5*, os tempos de i2 e i1 eram cerca de 10 vezes menores que os de p9 e 5 vezes menores que os de p3. Nos testes realizados na base real *virul fac* os tempos de i2 e i1 foram maiores que os de p9. O aumento dos tempos de i2 e i1 em relação a p9 foi atribuído ao aumento do tempo gasto com migração devido as sequências serem maiores.

No estudo feito com a variação dos intervalos de Migração, foi concluído que não existe a relação dos valores dos intervalos de Migração que obtém resultados bons ou ruins com as estratégias nem com as bases. Como a variação desses intervalos causa um grande impacto no resultado, deve-se executar uma fase de "*tuning*" para que os parâmetros mais adequados sejam escolhidos.

Como trabalhos futuros sugere-se:

- Buscar novas estratégias para o operador de junção das partes das sequências, pois o operador de junção atual reúne as partes que já estavam juntas antes de serem partidas, sem levar em consideração nenhum valor de aptidão. Seria interessante criar um operador que gerasse mais combinações, de forma que, quando as sequências fossem novamente cortadas, as populações de partes refletissem quais partes são melhores dentro do alinhamento inteiro.

- Implementar o modelo de Injeção de Ilhas proposto no algoritmo SAGA (seção 3.5.2). Dessa maneira, será possível a comparação dos resultados já existentes do SAGA, com os do SAGA no modelo Injeção de Ilhas.

- Utilizar comunicação assíncrona para a interação entre as ilhas, de modo a tentar reduzir o tempo de execução. Além disso, diminuir o tráfego das mensagens entre as ilhas, de forma a reduzir o *overhead* de comunicação.

- Implementar uma função de aptidão mais complexa, que guie o algoritmo para alinhamentos melhores. A função mais adequada seria a WSP (seção 3.4), pois ela é compatível com os operadores de mutação propostos nesse trabalho.

- Avaliar a estratégia proposta nessa dissertação com populações maiores e sequências mais longas.

- Possibilitar que os intervalos de Migração Externa sejam independentes entre as Super Ilhas. No modelo proposto, em que exitem Super Ilhas com processamentos diferentes umas das outras, poderia ser interessante a(s) Super Ilha(s) de um tipo enviarem indivíduos para a(s) outra(s) em intervalos diferentes. Por exemplo, a Super Ilha de Alta Resolução (SIAR) enviaria elementos para as Super Ilhas de Baixa Resolução (SIBR) no intervalo de 100 em 100 gerações, enquanto que as SIBR enviaria elementos para SIAR no intervalo de 50 gerações.

- Implementar a estratégia proposta em um modelo híbrido que envolva tanto troca de mensagem como compartilhamento de memória, de modo a explorar melhor as características dos *clusters multicore*, utilizando as bibliotecas OpenMP [5] e MPI.

Referências Bibliográficas

[1] Balibase. Disponível em http://bips.u-strasbg.fr/fr/Products/Databases/BAliBASE/, acessado em 04/08/2009.

[2] Jcreator versão 2.5. 2000. xinox software. Disponível em http://www.jcreator.com/, acessado em 11/07/2009.

[3] Medterms medical dictionary. Disponível em http://www.medterms.com/, acessado em 15/06/2009.

[4] Oma. Disponível em http://bibiserv.techfak.uni-bielefeld.de/oma/, acessado em 03/09/2008.

[5] Openmp. Disponível em http://openmp.org/wp/, acessado em 14/09/2008.

[6] Pfam. Disponível em http://pfam.sanger.ac.uk/family/browse, acessado em 04/08/2009.

[7] S. F. Altschul. Gap costs for multiple sequence alignment. *J. Theor. Biol.*, 138:297–309, 1989.

[8] S. Pascarella; P. Argos. A data bank merging related protein structures and sequences. *Protein Engineering*, 5(2):121–137, 1992.

[9] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[10] C. Bierwirth. A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spektrum*, 17:87–92, 1995.

[11] J.Heringa C. Notredame; D.G. Higgins. T-Coffee: a novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205–217, 2000.

[12] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Parallèles, Réseaux et Systèmes Répartis*, 10(2):141–171, 1998.

[13] E. Cantú-Paz. Implementing fast and flexible parallel genetic algorithms. *Pratical Handbook of Genetic Algorithms*, 3:65–84, 1999.

[14] E.G.M. Lacerda; A.C.P.L. Carvalho. Introdução aos algoritmos genéticos. In *Galvão, C.O., Valença, M.J.S. (orgs.) Sistemas inteligentes: aplicações a recursos hídricos e ciências ambientais*, pages 99–150, Porto Alegre: Ed. Universidade/UFRGS : Associação Brasileira de Recursos Hídricos (Coleção ABRH de Recursos Hídricos; 7.), 1999.

[15] R. Choudhury. Application of evolutionary algorithms for multiple sequence alignment. Project report, Stanford University, 2003.

[16] C. Darwin. *A Origem das Espécies e a Seleção Natural*. Hemus, 2000.

[17] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[18] E. G. M. de Lacerda; A. C. P. L. F. de Carvalho. Introdução aos algoritmos genéticos. In *C. Galvão; M. Valença. (Org.). Sistemas Inteligentes: Aplicações a Recursos Hídricos e Ciências Ambientais. 1 ed*, pages 99–148, Porto Alegre: Editora da Universidade Federal do Rio Grande do Sul, 1999.

[19] K. A. DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, MI, 1975. Dissertation Abstracts International 36(10), 5140B, University Microfilms Number 76-9381.

[20] D. Feng; R. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.*, 25:351–360, 1987.

[21] S.R. Eddy. Profile hidden markov models. *Bioinformatics*, 14:755–763, 1998.

[22] M. Gribskov; A. D. McLachlan; D. Eisenberg. Profile analysis: detection of distantly related proteins. *Proc Natl Acad Sci*, 84(13):4355–4358, 1987.

[23] S. F. Altschul; B. W. Erickson. Optimal sequence alignment using affine gap costs. *Journal of Molecular Biology*, 48(603):1–6, 1986.

[24] K. Chellapilla; G. B. Fogel. Multiple sequence alignment using evolutionary programming. In *Proceedings of the First Congress of Evolutionary Computation (CEC-1999)*, pages 445–452. IEEE Press, 1999.

[25] K. Chellapilla; G. B. Fogel. Multiple sequence alignment using evolutionary programming. *Congress on Evolucionary Computation (CEC-1999)*, 1999.

[26] J. D. Thompson; D. G. Higgins; T. J. Gibson. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, 22:4673–4680, 1994.

[27] D. E. Goldberg. *Genetic Algorithms in Search, Optimisation, and Machine Learning*. Addison-Wesley, 1989.

[28] M. Babbar; B. S. Minsker; D. Goldberg. A multiscale island injection genetic algorithm for optimal groundwater remediation design. 2002. American Society of Civil Engineers (ASCE) Environmental & Water Resources Institute (EWRI) 2002 Water Resources Planning & Management Conference, Roanoke, VA.

[29] O. Gotoh. Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *Journal of Molecular Biology*, 264:823–838, 1996.

[30] D. Gusfield. *Algorithms on strings, trees, and sequences: Computer science and computational biology.* Cambridge University Press, Cambridge, 1997.

[31] D. Whitley; S. Rana; R. B. Heckendorn. Island model genetic algorithms and linearly separable problems. In *Proceedings of AISB Workshop on Evolutionary Computation, volume 1305 of LNCS*, pages 109–125. Springer Verlag, 1997.

[32] S. Henikoff; J. G. Henikoff. Amino acid substitution matrices from protein blocks. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 89, pages 10915–10919. Nacional Academy of Sciences, 1992.

[33] C. Notredame; D. G. Higgins. SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research*, 24(8):1515–1524, 1996.

[34] C. Notredame; L. Holm; D.G. Higgins. COFFEE: an objective functions for multiple sequence alignments. *Bioinformatics*, 14(5):407–422, 1998.

[35] D. G. Higgins. Clustal v: Multiple alignment of dna and protein sequences. In *Computer Analysis of Sequence Data*, pages 307–318. Humana Press, 1994.

[36] D. S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24:664–675, 1977.

[37] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[38] D. A. Goldberg; K. Deb; J. Horn. Massive multimodality, deception, and genetic algorithms. In *PPSN*, pages 37–48, 1992.

[39] K. Karplus; B. Hu. Evaluation of protein multiple alignments by sam-t99 using the balibase multiple alignment test set. *Bioinformatics*, 17:713–720, 2001.

[40] L. Wang; T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.

[41] J. Kececioglu. The maximum weight trace problem in a multiple sequence alignment. *Lecture Notes in Computer Science*, 684:106–119, 1993.

[42] V. Sundararajan; A. S. Kolaskar. Distributed genetic algorithms on param for conformational search. In S. Seetharama Iyengar, editor, *Computer Modeling and simulations of Complex Biological systems*, pages 16–25. CRC Press, 1998.

[43] R. Thomsen; G. B. Fogel; T. krink. A clustal alignment improver using evolucionary algorithms. *Proceedings of the Fourth Congress on Evolucionary Computation (CEC-2002)*, 1:121–126, 2002.

[44] S. Chen; C. Lin. Multiple dna sequence alignment based on genetic algorithms and divide-and-conquer techniques. *International Journal of Applied Science and Engineering*, 3(8):89–100, 2005.

[45] S. F. Altschul; R. J. Carroll; D. J. Lipman. Weights for data related by a tree. *Journal of Molecular Biology*, 207:647–653, 1989.

[46] K. Yan; Y. Lu. Sidelobe reduction in array-pattern synthesis using geneticalgorithm. *Antennas and Propagation, IEEE Transactions*, 45(7):1117–1122, 1997.

[47] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, third edition, 1994.

[48] K. Katoh; K. Misawa; K. Kuma; T. Miyata. MAFFT: a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic Acids Research*, 30:3059–3066, 2002.

[49] L. Abdesslem; M. Soham; B. Mohamed. Multiple sequence alignment by quantum genetic algorithm. In *Proceedings 20th IEEE International Parallel; Distributed Processing Symposium*, pages 360–368, 2006.

[50] D. W. Mount. *Bioinformatics: Sequence and Genome Analysis*. Laboratory Press, Cold Spring Harbor, 2004.

[51] C. Notredame. Recent progress in multiple sequence alignment: a survey. *Pharmacogenomics*, 3:131–144, 2002.

[52] F. Armougom; S. Moretti; O. Poirot; S. Audic; P. Dumas; B. Schaeli; V. Keduas; C. Notredame. Expresso: automatic incorporation of structural information in multiple sequence alignments using 3d-coffee. *Nucleic Acids Research*, 34:604–608, 2006.

[53] M. O. Dayhoff; R. M. Schwartz; B. C. Orcutt. A model pf evolucionary change in proteins. In *Atlas of Protein Sequence and Structure*, volume 5, pages 345–352, Washinton, DC, 1978. M. O. Dayoff.

[54] J. D. Thompson; F. Plewniak; O. Poch. BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15:87–88, 1999.

[55] T. Strachan; A. P. Read. *Human Molecular Genetics 2.* John Wiley & Sons, 1999.

[56] L. J. Eshelman; D. J. Shaffer. Real-coded genetic algorithms and interval-schemata. In *WHITLEY , D. L. (ed). Foundations of Genetic Algorithms 3*, pages 187–203, SanMateo , CA, 1992.

[57] L. A. Anbarasu; P. Narayanasamy; V. Sundararajan. Multiple sequence alignment using parallel genetic algorithms. *Lecture Notes in Computer Science*, 1585:130–137, 1999.

[58] L.A. Anbarasu; P. Narayanasamy; V. Sundararajan. Multiple molecular sequence alignment by island parallel genetic algorithm. *Current Science*, 78(7):858–863, 2000.

[59] G. Syswerda. The application of genetic algorithms to resource scheduling. *Proceedings from the Fourth International Conference on Genetic Algorithms*, 1:502–508, 1991.

[60] T. F. Smith; M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[61] D. Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, 1989. Morgan Kaufman.

[62] C. Zhang; A. K. C. Wong. A genetic algorithm for multiple molecular sequence alignment. *Computer Applications in the Biosciences*, 13:565–581, 1997.

[63] S. B. Needleman; C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, pages 443–453, 1970.

[64] H. D. Nguyen; I. Yoshihara; K. Yamamori; M. Yasunaga. Aligning multiple protein sequences by parallel hybrid genetic algorithm. *Genome Informatics*, 13:123–132, 2002.

[65] H. D. Nguyen; I. Yoshihara; K. Yamamori; M. Yasunaga. A parallel hybrid genetic algorithm for multiple protein sequence alignment. *Evolutionary Computation*, 1:309–314, 2002.

[66] N. Morad; A. M. S. Zalzala. Genetic algorithms in integrated process planning and scheduling. *Journal of Intelligent Manufacturing*, 1999.