2
Contexto e Motivações

Existem duas principais famílias de algoritmos que realizam comparações de biosseqüências, a FASTA (Pearson, 1991) e a BLAST (Altschul et al., 1990). Estas ferramentas realizam alinhamentos locais para encontrar regiões nas quais duas seqüências possuem alto grau de similaridade, sendo diferente do alinhamento global, que busca alinhar pares de seqüências em toda a extensão. Uma das vantagens da realização de alinhamentos locais é a possibilidade de descoberta de domínios conhecidos nas proteínas usadas como consulta ou codificadas por genes usados como consulta. Os domínios de proteínas são blocos de suas extensões com composições e funções específicas, sendo que domínios similares, mesmo em proteínas de funções diferentes, têm eles próprios funções semelhantes. Portanto, pode-se estimar a função de uma proteína por sua semelhança a outras proteínas.

As ferramentas da família BLAST são as mais utilizadas pelos pesquisadores para comparação de biosseqüências e existem diversos *sites* que disponibilizam os algoritmos para os usuários, podendo ocorrer acessos simultâneos. O desempenho é um fator muito importante para estas ferramentas e pequenas melhoras nestas podem trazer grandes benefícios.

Durante a realização deste trabalho, foi necessário estudar diversas características do BLAST. Primeiramente, foi realizado um estudo da família BLAST, seu algoritmo principal e suas variações, para possibilitar a identificação de melhorias possíveis.

A estratégia utilizada neste trabalho inclui o provimento ao BLAST de bancos de dados com conteúdos diferentes do banco original sem, no entanto, alterar o resultado da sua execução. Para verificar esta possibilidade, foram necessários estudos dos diferentes formatos existentes dos bancos de dados do BLAST. Esse conhecimento é importante para a implementação da ferramenta e

para decisões de como torná-la menos dependente dos formatos do banco, funcionando com diversas versões e implementações do BLAST.

Em seguida, foi feita uma análise dos padrões de leitura do banco feitos pelo BLAST. Esse estudo complementa a análise do algoritmo do BLAST e confirma a necessidade de melhoria do modo como o banco é acessado. Ainda por cima, é essencial para possibilitar a identificação em tempo de execução da etapa do algoritmo que está sendo executada por um processo BLAST.

Finalmente, foram estudados os diferentes trabalhos realizados para melhorar o desempenho do BLAST e as propostas existentes. Os resultados principais dos estudos realizados são apresentados nas seções a seguir.

2.1. A Família BLAST

O programa do BLAST implementa uma heurística para o alinhamento local de biosseqüências. Existem diferentes implementações do BLAST, sendo mais conhecidos o NCBI BLAST e o WU-BLAST.

O NCBI BLAST é uma implementação mantida pelo NCBI (National Center for Biotechnology Information), nos Estados Unidos. O NCBI mantém bancos de dados públicos, conduz pesquisa em biologia computacional, desenvolve ferramentas de *software* para a análise de dados genômicos e dissemina informações biomédicas. O código e os executáveis do NCBI BLAST estão disponíveis em domínio público em NCBIa (2006). Na data de escrita deste trabalho, a versão corrente do NCBI BLAST é a 2.2.14.

O WU-BLAST é a implementação da estratégia mantida pela Universidade de Washington, nos EUA. A universidade mantém em domínio público apenas o código da versão 1.4 de seu programa e alguns executáveis mais recentes com funcionalidade reduzida. Estes podem ser encontrados em WU (2006). A versão completa da implementação está sujeita a uma política de licenciamento. Na data de escrita deste trabalho, a versão corrente do WU-BLAST é a 2.0.

2.1.1.
Principais Características

O BLAST compara seqüências de entrada com todas as seqüências de um banco de dados, realizando alinhamentos locais. As seqüências de entrada são enviadas pelo usuário no formato FASTA e podem ser tanto de nucleotídeos, quanto de aminoácidos. Os bancos de dados lidos pelo BLAST possuem um formato específico, sendo formados, de modo geral, por 3 arquivos: um de seqüências, um de anotações associadas às seqüências e um com índices relacionando as seqüências às suas anotações (WU, 2006). As anotações das seqüências incluem informações sobre as mesmas, como seus números identificadores e funcionalidades principais. São usadas ferramentas para gerar os bancos de dados a partir dos mesmos nos formatos FASTA ou ASN.1.

O algoritmo básico do BLAST possui três etapas. Na primeira etapa são criadas pequenas seqüências de tamanho fixo (normalmente de 3 ou 4 letras), denominadas palavras, que possuem grande similaridade com partes da seqüência de entrada. Na segunda etapa, são encontrados todos os casamentos exatos destas palavras com as seqüências do banco de dados. Finalmente, na terceira etapa, os casamentos exatos encontrados na segunda etapa são estendidos, realizando-se alinhamentos locais. Uma descrição mais detalhada do funcionamento de BLAST é apresentada no apêndice A deste documento.

Observa-se que, na segunda etapa do algoritmo, por razões de completude, todo o arquivo de seqüências é varrido. Esta é também a etapa na qual há um maior número de leituras feitas do disco para a memória. Já que o BLAST lê diretamente de arquivos do sistema operacional, não utilizando um SGBD, a leitura das seqüências do banco durante a segunda etapa do algoritmo é feita de maneira ineficiente em algumas situações que são muito comuns. A leitura ineficiente ocorre quando o arquivo de seqüências não pode ser mantido inteiramente na memória e diversos processos são executados ao mesmo tempo, pois a execução de cada processo deverá fazer com que parte do banco de seqüências seja lido novamente do disco para a memória durante a segunda etapa do algoritmo.

A segunda etapa do algoritmo BLAST possui também uma característica importante: como o objetivo é analisar todas as seqüências do banco de dados a procura de similaridades, não é importante a ordem em que é realizada a comparação da seqüência de consulta com cada seqüência do banco. Logo, um processo BLAST pode começar sua comparação por qualquer seqüência do banco desde que, ao final da execução da segunda etapa do algoritmo, todas as seqüências tenham sido comparadas.

2.1.2. Subprogramas

O programa BLAST se divide em cinco subprogramas, cujas diferenças básicas estão no tipo de comparação que o usuário deseja fazer:

- BLASTP (proteínas x proteínas): Utiliza uma seqüência de aminoácidos como entrada, a qual é comparada com um banco de dados de proteínas. Esse tipo de BLAST é muito utilizado quando se tem uma proteína e deseja-se saber se existem, em outros organismos, proteínas similares. A vantagem em se utilizar o BLASTP é que pode-se comparar proteínas de organismos muito distantes evolutivamente, já que as matrizes de pontuação usadas no alinhamento de seqüências de proteínas levam em conta fatores evolutivos.
- BLASTN (nucleotídeos x nucleotídeos): Utiliza uma seqüência de nucleotídeos como entrada, a qual é comparada com um banco de dados de nucleotídeos. O BLASTN é muito utilizado para descobrir seqüências de nucleotídeos muito conservadas, desde genes a pequenas seqüências que servem como marcadores moleculares. Como a matriz pontuação usada no alinhamento de seqüências de nucleotídeos é pouco flexível, o BLASTN deve ser usado para comparar seqüências de organismos próximos biologicamente.
- BLASTX (nucleotídeos x proteínas): Utiliza uma seqüência de nucleotídeos como entrada, a qual é traduzida em seqüências de aminoácidos, que são comparadas com um banco de dados de proteínas.

São necessários 3 nucleotídeos para codificar um aminoácido e a tradução do DNA em proteínas é feita por meio de um dos seus 2 filamentos complementares, sendo que nucleotídeos *a* correspondem a *t* e nucleotídeos *g* correspondem a *c* nos filamentos complementares. Logo, há 6 maneiras diferentes de traduzir seqüências de nucleotídeos em seqüências de proteínas, de acordo com 6 quadros de leitura. Cada uma das 6 seqüências de proteínas traduzidas é comparada com o banco de proteínas de maneira semelhante ao BLASTP. O BLASTX é usado para determinar o quadro de leitura no qual uma determinada seqüência de nucleotídeos deve ser traduzida em proteínas, além de dar valor biológico às possíveis proteínas.

– TBLASTN (proteínas x nucleotídeos): Utiliza uma seqüência de aminoácidos como entrada, a qual é comparada com um banco de dados de nucleotídeos. As seqüências de nucleotídeos do banco são traduzidas nos seis possíveis quadros de leitura. O TBLASTN é usado para encontrar genes novos que ainda não foram descritos antes, devido ao grande numero de proteínas desconhecidas, principalmente as de pequeno tamanho.

– TBLASTX (nucleotídeos x nucleotídeos): Utiliza uma seqüência de nucleotídeos como entrada, a qual é traduzida em seqüências de aminoácidos nos 6 quadros de leitura. As seqüências de aminoácidos são comparadas com um banco de dados de nucleotídeos também traduzidos em aminoácidos nos 6 quadros de leitura. O TBLASTX é usado principalmente para dar valor biológico a seqüências de nucleotídeos desconhecidas.

2.2. Formatos dos Bancos de Dados

Os bancos de dados do BLAST podem possuir diferentes formatos, dependendo da versão utilizada, do tipo de dados do banco (aminoácidos ou nucleotídeos) e dos parâmetros de formatação selecionados. De modo geral, os

bancos são formados por um arquivo de seqüências compactado, um arquivo de anotações associadas às seqüências e um arquivo de índices relacionando as seqüências com suas anotações. Os formatos desses arquivos variam um pouco com as diferentes versões do BLAST, sendo que o banco de dados do NCBI BLAST está atualmente na versão 4. Entretanto, nem todas as modificações nos formatos do banco são informadas pelos desenvolvedores (UBiC, 2006), o que pode tornar custosa a identificação do formato real do banco.

Neste trabalho, os formatos de bancos de dados mais importantes a serem estudados são os de proteínas das versões mais recentes do NCBI BLAST (versão 2.0 em diante) e os da última versão do WU-BLAST com código aberto (versão 1.4), conforme explicado em WU (2006). A versão mais recente do WU-BLAST não possui código aberto e o formato do seu banco de dados não está bem documentado, logo não foi levada em conta no presente trabalho.

2.2.1. O Banco de Dados do NCBI BLAST

O programa *formatdb* é utilizado para gerar o banco de dados do NCBI BLAST a partir de um arquivo em formato FASTA. No banco de proteínas, o arquivo de seqüências é criado com terminação *.psq*, o de anotações, com terminação *.phr* e o de índices, com terminação *.pin*. Dependendo dos parâmetros usados com o *formatdb*, o banco de dados pode conter também outros arquivos.

Na versão 2.0 do NCBI BLAST, o arquivo de seqüências é formado pelas seqüências na ordem na qual estas aparecem no arquivo FASTA, sendo que cada caractere correspondente a um aminoácido é substituído por um código binário, conforme mostrado na tabela 1. Cada caractere é representado em 1 byte. O caractere "-" (código 0) é usado como separador de seqüências, além de ser o primeiro e o último caractere do arquivo.

Caractere	Código	Caractere	Código
-	0	M	12
A	1	N	13
B	2	P	14
C	3	Q	15
D	4	R	16
E	5	S	17
F	6	T	18
G	7	V	19
H	8	W	20
I	9	X	21
K	10	Y	22
L	11	Z	23
		*	24

Tabela 1: Códigos dos caracteres de aminoácidos do NCBI 2.0

O arquivo de anotações é formado pelas anotações das seqüências, uma depois da outra, na ordem na qual estas aparecem no arquivo FASTA. As anotações são armazenadas no formato ASN.1 binário.

O arquivo de índices possui, primeiramente, informações sobre o banco de dados, como a versão do programa *formatdb* usado para gerá-lo, o tipo de dado (nucleotídeos ou aminoácidos), o nome do banco, a data de criação, o número de seqüências, etc. Em seguida, são armazenadas as posições de início de cada anotação no arquivo de anotações. Finalmente, são armazenadas as posições de início de cada seqüência no arquivo de seqüências.

É possível gerar o banco de dados com 2 ou 4 arquivos adicionais se for desejável, por exemplo, obter o arquivo de saída do BLAST em formato ASN.1. Um arquivo é o de diretório textual, com terminação *.psd*, que possui todos os diferentes identificadores únicos, em formato texto, das seqüências do arquivo FASTA, seguidos do número de ordem da respectiva seqüência no arquivo FASTA em relação às outras seqüências do arquivo. Este arquivo possui formato textual porque a informação de identificador único de uma seqüência inclui, além de um número, a informação sobre o banco de dados no qual este número é único (por exemplo, “pir||a64226”). Cada seqüência pode ter mais de um identificador único se esta estiver presente em bancos de dados diferentes. O outro arquivo gerado, com terminação *.psi*, possui um índice para acelerar buscas no arquivo de diretório textual.

Os outros 2 arquivos são o de diretório numérico, com terminação *.pnd*, e o de índice para buscas nesse diretório, com terminação *.pni*. O arquivo de diretório numérico é semelhante ao de diretório textual. Entretanto, este possui apenas informações sobre identificadores únicos relativos ao GenBank, e os dados do arquivo são todos numéricos.

2.2.2. O Banco de Dados do WU-BLAST

O programa *setdb* é utilizado para gerar o banco de dados do WU-BLAST 1.4 a partir de um arquivo em formato FASTA. No banco de proteínas, o arquivo de seqüências é criado com terminação *.bsq*, o de anotações, com terminação *.ahd* e o de índices, com terminação *.atb*.

O arquivo de seqüências do WU-BLAST 1.4 é muito semelhante ao do NCBI BLAST 2.0. A única diferença entre ambos os formatos são os códigos binários para os quais são convertidos os caracteres correspondentes a aminoácidos. Os códigos de conversão são mostrados na tabela 2.

Caractere	Código	Caractere	Código
-	0	K	12
A	1	M	13
R	2	F	14
N	3	P	15
D	4	S	16
C	5	T	17
Q	6	W	18
E	7	Y	19
G	8	V	20
H	9	B	21
I	10	Z	22
L	11	X	23
		*	24

Tabela 2: Códigos dos caracteres de aminoácidos do WU-BLAST 1.4

O arquivo de anotações, assim como o da versão 2.0 do NCBI BLAST, é formado pelas anotações das seqüências, uma depois da outra, na ordem na qual estas aparecem no arquivo FASTA. A diferença entre ambos os formatos é que, neste caso, as anotações são armazenadas no formato textual, exatamente como

aparecem no arquivo FASTA. O caractere “>” é usado como separador de anotações e não há outros caracteres “>” nas anotações.

O arquivo de índices possui, primeiramente, informações sobre o banco de dados, que incluem o tipo de dado do banco (nucleotídeos ou aminoácidos), o nome do banco, o número de seqüências e o tamanho da maior seqüência do banco. Em seguida, são armazenadas as posições de início de cada seqüência no arquivo de seqüências. Finalmente, são armazenadas as posições de início de cada anotação no arquivo de anotações.

2.3. O Acesso do BLAST ao Banco de Dados

Nesta seção são apresentados os acessos do NCBI BLAST e do WU-BLAST ao banco de dados de proteínas. O banco de dados usado na comparação é o *ecoli.aa*, que pode ser obtido em NCBIa (2006). Foram escolhidas como seqüências de entrada seqüências pertencentes ao mesmo banco, para garantir que o BLAST encontrasse similaridades.

2.3.1. NCBI BLAST com 1 Seqüência de Entrada

A figura 1 apresenta as posições dos arquivos do banco de dados de proteínas lidas pelo NCBI BLAST durante sua execução com 1 seqüência de entrada. No eixo *x* são indicados os números das requisições de leitura feitas pelo BLAST. Quase todas as requisições são de 4096 *bytes*, que é o tamanho de página do sistema operacional, exceto algumas requisições de leitura do arquivo de índices. Verifica-se pelo gráfico que, inicialmente, o BLAST lê o arquivo de índices. Em seguida, todo o arquivo de seqüências é lido. Ao final da execução, tanto o arquivo de seqüências quanto o de anotações são lidos em posições aleatórias, porém semelhantes em ambos os arquivos. O arquivo de índices também é lido 2 vezes ao final da execução.

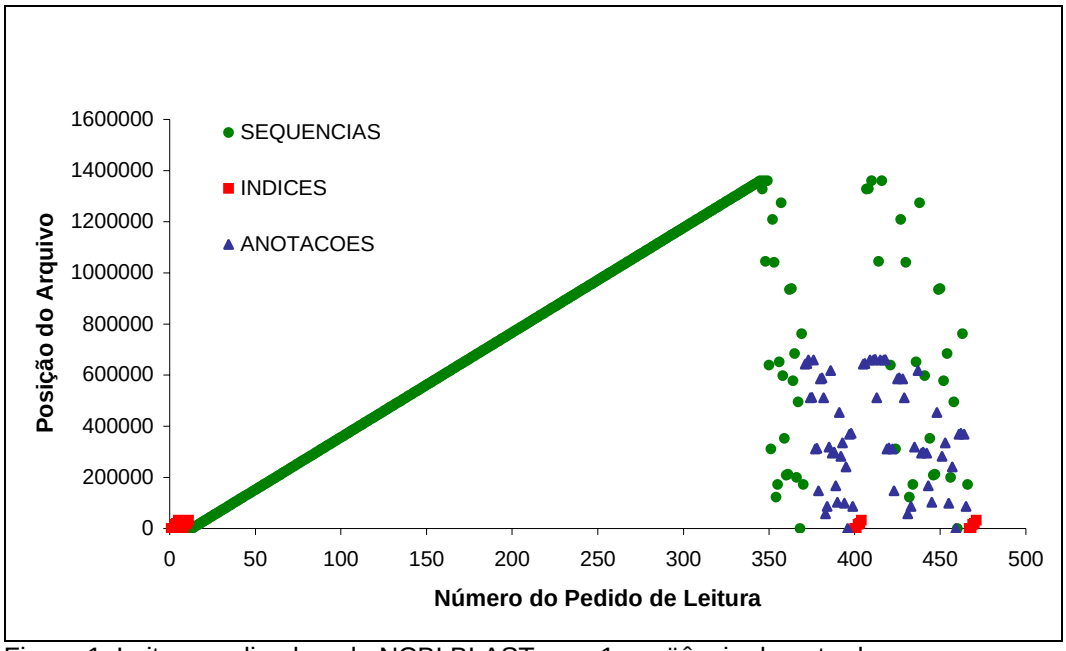

Figura 1: Leitura realizada pelo NCBI BLAST com 1 seqüência de entrada

Ao analisar o código do BLAST, verificou-se que este começa a ler o banco apenas na segunda fase do algoritmo, na qual são encontrados os casamentos das seqüências do banco com as palavras selecionadas na primeira etapa. Entretanto, a varredura do início ao fim do arquivo de seqüências, mostrada no gráfico, corresponde ao mesmo tempo à segunda e à terceira etapa do algoritmo. Isso ocorre porque, ao contrário do esperado, a terceira etapa não é executada apenas ao final da segunda etapa, mas ocorre ao mesmo tempo. Ou seja, quando cada casamento de palavras é encontrado na segunda etapa, este é estendido logo em seguida. Finalmente, foi possível verificar que os pedidos de leitura de posições aleatórias dos arquivos de seqüências e anotações correspondem à etapa de geração do relatório do BLAST, com os resultados obtidos nas comparações, no qual constam as seqüências do banco mais similares à seqüência de entrada e suas respectivas anotações.

Uma importante observação a ser feita é a de que a maior parte das requisições de leitura ocorre na etapa de leitura seqüencial do arquivo de seqüências e as requisições restantes ocorrem quase todas durante a geração do relatório. Na verdade, a proporção entre a quantidade de leituras da etapa de varredura seqüencial e a quantidade total de leituras depende principalmente do tamanho do banco e dos parâmetros de entrada usados, podendo resultar na seleção de muitas ou poucas seqüências para o relatório. Na prática, espera-se que o biólogo utilize parâmetros bem restritivos para que seja retornado no relatório

um percentual pequeno das seqüências do banco. Nesse caso, a etapa de varredura do arquivo de seqüências será bem maior que a de geração do relatório.

2.3.2.
NCBI BLAST com 5 Seqüências de Entrada

A figura 2 apresenta as posições dos arquivos do banco de dados de proteínas lidas pelo NCBI BLAST durante sua execução com 5 seqüências de entrada.

Verifica-se que, quando há um maior número de seqüências de entrada, o BLAST realiza leituras do banco de dados de maneira semelhante ao caso onde há apenas 1 seqüência de entrada. Inicialmente o arquivo de índices é lido 2 vezes, em seguida o arquivo de seqüências é lido do início ao fim. Ao final, os arquivos de seqüências e anotações são lidos em posições aleatórias e o arquivo de índices é lido novamente 2 vezes.

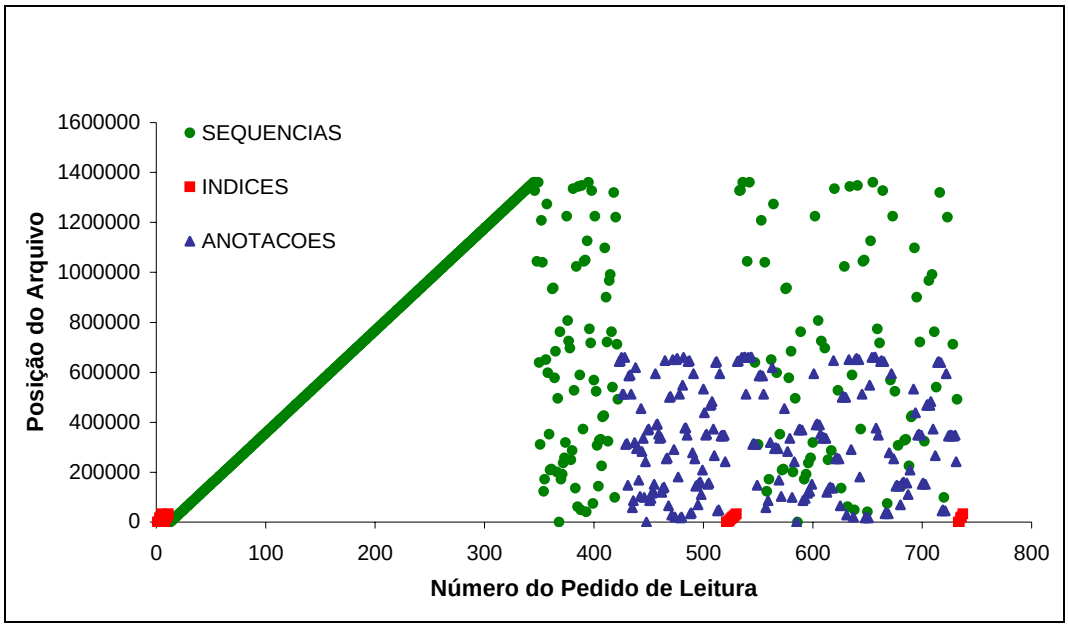

Figura 2: Leitura realizada pelo NCBI BLAST com 5 seqüências de entrada

2.3.3.
WU-BLAST com 1 Seqüência de Entrada

A figura 3 apresenta as posições dos arquivos do banco de dados de proteínas lidas pelo WU-BLAST durante sua execução com 1 seqüência de entrada. Verifica-se que, inicialmente, o arquivo de índices é lido do início ao fim. Em seguida, o arquivo de seqüências é lido do início ao fim. Ao final, os arquivos de seqüências e anotações são lidos em posições aleatórias, porém semelhantes em ambos os arquivos.

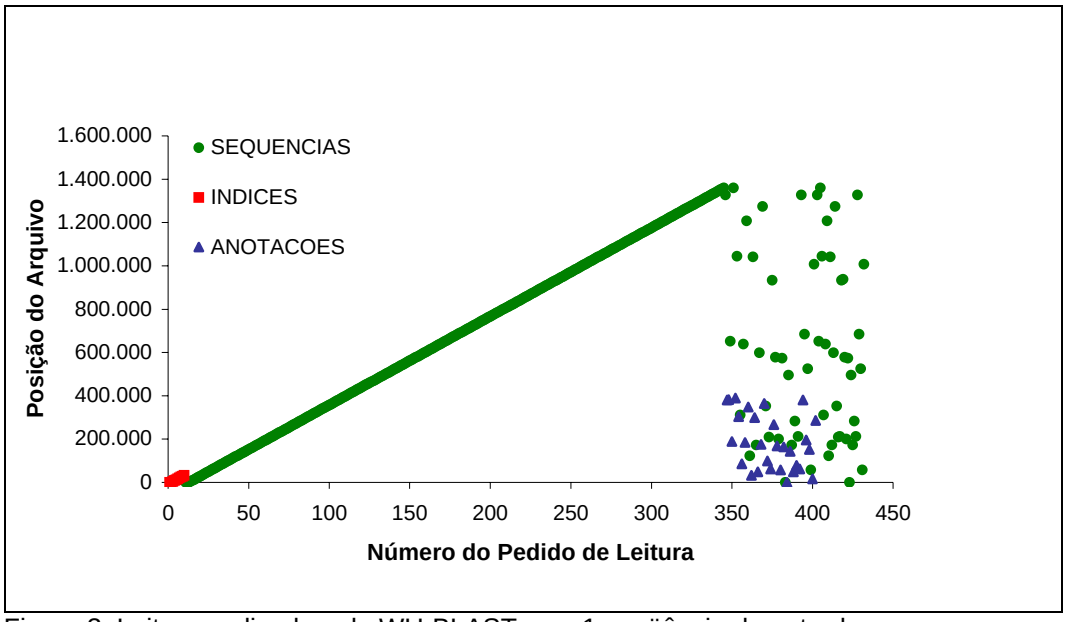

Figura 3: Leitura realizada pelo WU-BLAST com 1 seqüência de entrada

Cada processo do WU-BLAST compara apenas uma seqüência de entrada com o banco de dados. Se o arquivo com as seqüências de consulta possuir mais de uma seqüência, o WU-BLAST utilizará apenas a primeira seqüência e ignorará as outras.

2.4.
Trabalhos de Melhoria do BLAST

Por atingir resultados satisfatórios em um tempo relativamente curto, o BLAST se tornou um dos mais populares programas para alinhamento local de

biosseqüências. Esta grande aceitação levou os criadores do BLAST a desenvolverem evoluções do algoritmo para melhorar sua velocidade e sua sensibilidade, e algumas variantes do programa para aplicações específicas.

Algumas das evoluções mais importantes do algoritmo do BLAST são a possibilidade de executar alinhamentos locais com a introdução de buracos e a limitação da extensão de alinhamentos para regiões de seqüências onde há pelo menos 2 *hits* (casamentos de palavras) próximos. Estas duas melhorias são descritas em Altschul et al. (1997).

Algumas variantes do BLAST desenvolvidos pelo NCBI são o PSI-BLAST e o PHI-BLAST (NCBIa, 2006). Através do PSI-BLAST, é possível detectar relações muito distantes entre biosseqüências, já que a pontuação de cada aminoácido pode ser mudada e muitas iterações podem ser realizadas. Isto permite que o usuário inclua seqüências específicas em sua busca, para construir o "perfil" para a iteração seguinte. O PHI-BLAST é semelhante ao BLASTP, porém permite que a seqüência de entrada contenha padrões em vez de uma única seqüência. Isto pode ser útil quando se deseja procurar proteínas que contenham um determinado domínio. Mais recentemente, estas duas variantes do BLAST se juntaram no programa BLASTPGP.

Uma variante do BLAST para o acesso a bancos de dados de nucleotídeos é o MEGABLAST (NCBIa, 2006), que usa um algoritmo guloso para a busca no banco de dados concatenando diversas seqüências de entrada. O MEGABLAST deve ser usado para comparar seqüências de nucleotídeos muito similares e objetiva diminuir o tempo de leitura do banco de dados.

Outras melhorias da estratégia do BLAST foram desenvolvidas por pesquisadores. Em Rosa et al. (2006), foi implementada no BLAST a leitura do banco compactado usando um algoritmo específico, objetivando diminuir o tamanho do banco a ser lido e melhorar o desempenho do BLAST. Alguns trabalhos procuram aumentar o desempenho do BLAST através de melhorias no algoritmo deste (Cameron et al., 2004) ou do uso de técnicas de processamento paralelo (Costa et al., 2003) (Wang et al. 2003) (Darling et al. 2003) (Lin et al., 2005). Alguns trabalhos propõem, para o mesmo fim, uma gerência mais eficaz do acesso ao banco de dados de seqüências (Lemos et al., 2003) (Mauro et al., 2004). Estes últimos serão descritos nas seções a seguir.

2.4.1.
Gerenciamento de *Buffer* para o BLAST

Devido às características do BLAST descritas na seção 2.1, foi sugerida em Lemos et al. (2003) uma estratégia de gerenciamento de *buffer* para o BLAST. A estratégia deve ser aplicada na segunda etapa do algoritmo, na qual o banco de dados é varrido à procura de *hits*. A vantagem na sua utilização ocorre nos casos em que o arquivo de seqüências lido pelo BLAST não pode ser mantido inteiramente na memória e diversos processos lêem ao mesmo tempo do mesmo banco.

Na estratégia proposta, são usadas estruturas de armazenamento de seqüências na memória denominadas anéis. Um anel consiste em um *buffer* na memória para o qual as seqüências do banco vão sendo carregadas aos poucos. O anel é lido pelos vários processos BLAST de maneira compartilhada, sendo que as atualizações de partes do anel ocorrem quando todos os processos em execução sendo atendidos já tiverem consumido a informação disponível. A figura 4 ilustra o acesso ao anel. Neste exemplo, quatro processos BLAST em execução concorrente (B1, B2, B3 e B4) acessam as páginas do anel em memória de maneira compartilhada. A figura mostra também a ordem cíclica na qual as páginas do anel são acessadas, sendo que, após lerem a última posição do anel, os processos continuarão a leitura a partir da primeira posição novamente.

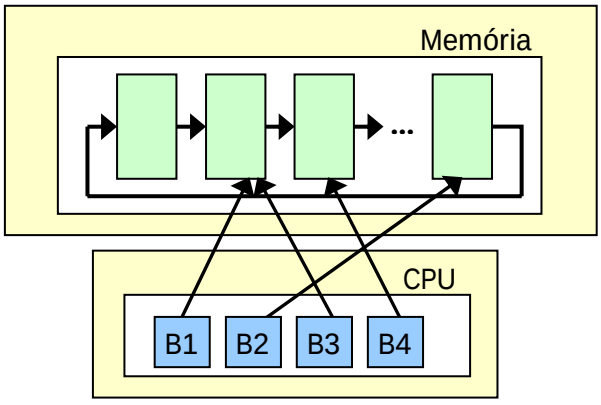

Figura 4: Acesso dos processos BLAST às páginas do anel em memória

Supondo a existência de apenas um anel na memória, de *n* páginas, e apenas um banco de dados de seqüências, a estratégia consiste em fazer com que os processos leiam as seqüências do anel de maneira compartilhada. Inicialmente, as primeiras *n* páginas com as seqüências são transferidas do banco de dados para o anel e posicionadas ordenadamente. Os processos lêem as páginas do anel em ordem, começando pela primeira. Sempre que todos os processos já tiverem lido a mesma página esta pode ser substituída pela próxima página a ser transferida do banco para o anel. Deste modo, quando os processos atingirem o final do anel, estes devem continuar a leitura a partir do início do mesmo, que conterá a próxima página, desde que esta posição do anel já tenha sido atualizada. Quando a última seqüência do banco for transferida para o anel, a transferência das seqüências para o *buffer* será reiniciada a partir da primeira seqüência do banco.

Alguns processos executarão mais rapidamente que outros. A velocidade de execução do processo do BLAST varia com o tamanho da seqüência de consulta e a similaridade desta com as seqüências do banco de dados. Se um processo tentar ler de uma posição do anel que ainda não foi atualizada com a página requisitada porque algum processo mais atrasado ainda está lendo desta posição, este terá que esperar a liberação e atualização da página do anel.

A qualquer momento um novo processo pode iniciar sua execução. Já que a ordem na qual as seqüências são lidas não é importante, este processo começará a leitura das seqüências a partir das que estiverem disponíveis no anel. Como as seqüências do banco são transferidas novamente para o anel após a leitura da última seqüência do banco, o novo processo lerá também todas as seqüências anteriores à seqüência inicialmente considerada, completando a leitura do banco. Esta é a grande melhora em relação às políticas de gerenciamento de *buffer* oferecidas pelos sistemas operacionais, pois são usados conhecimentos sobre a maneira como o BLAST acessa e utiliza o banco de dados para otimizar sua leitura do banco, sendo que a ordem das seqüências lidas pelos processos nem sempre será igual.

Ao usar o gerenciamento de *buffer* com um anel, todos os processos lendo de um mesmo anel executarão com velocidade aproximadamente igual à do processo mais lento, pois este não poderá estar mais de um ciclo de leitura do anel atrás do processo mais rápido. Para melhorar o desempenho dos processos,

múltiplos anéis podem existir, sendo que processos com velocidades semelhantes devem ser alocados ao mesmo anel.

Esta estratégia de uso eficiente do *buffer* é interessante para ser oferecida junto com o BLAST, pois além de possibilitar o *prefetching*, permite que vários processos compartilhem do mesmo *buffer*, diminuindo assim o tempo gasto em leituras do disco para a memória.

2.4.2. Gerenciamento de *Buffer* Não-Intrusivo

Há dois modos de implementar o gerenciamento de *buffer* para o BLAST: de maneira intrusiva no código deste e de maneira não-intrusiva. A maneira intrusiva de implementar é através da substituição, no código, de cada chamada às funções de leitura de seqüências por outras funções que se comunicam com um processo provedor de dados que irá realizar o gerenciamento de *buffer*. A maneira não-intrusiva de implementar é criando uma ferramenta que se comunica com o BLAST sem exigir a modificação do código do mesmo.

A implementação intrusiva no código é uma solução mais simples e, por tratar de apenas uma implementação do BLAST, pode trazer um melhor desempenho levando-se em conta características específicas do mesmo. Por outro lado, uma implementação não-intrusiva no código pode ser genérica o suficiente para funcionar com diversas versões e implementações do BLAST ou exigir poucas modificações no código da ferramenta para funcionar com uma nova versão do BLAST. A solução intrusiva no código foi usada nesse mesmo grupo de pesquisa para acrescentar o gerenciamento de buffer ao WU-BLAST 1.4, trazendo grandes melhoras no seu desempenho. Entretanto, o WU-BLAST 1.4 é pouco usado pelos cientistas atuais, por ser uma versão ultrapassada do BLAST.

Para a implementação do gerenciamento de buffer de maneira não-intrusiva no código, foi sugerida, em Mauro et al. (2004), a utilização de um *driver* de dispositivo que simule o funcionamento dos arquivos do banco de dados e realize a comunicação entre os processos BLAST e o processo provedor. Um *driver* de dispositivo é uma camada de *software* do sistema operacional que possibilita a

comunicação de aplicativos com dispositivos de *hardware* e *software*, escondendo a maneira como é realizada a comunicação direta com os dispositivos. A idéia proposta em Mauro et al. (2004) consiste em substituir os arquivos do banco de dados do BLAST por arquivos especiais associados a um *driver* de dispositivo de caractere. Ao executar funções de abertura e leitura destes arquivos, o BLAST chama na realidade as funções implementadas pelo *driver*, que realizam a comunicação com o processo provedor de banco de dados.

Para testar a comunicação entre os processos, foi implementada em Mauro et al. (2004) uma versão limitada do *driver*. Nesta versão, o arquivo de seqüências do banco é substituído por um arquivo especial. Assim que o provedor recebe uma requisição de página, esta é lida diretamente do arquivo de seqüências e enviada como resposta através da função de escrita do *driver*. Ou seja, ainda não há um gerenciamento de memória propriamente dito, mas uma substituição da leitura direta do arquivo de índices pela leitura usando um *driver* e um processo provedor. Foi possível verificar, através de testes, que não houve grande piora no desempenho do BLAST devido à utilização de um *driver* e um processo provedor durante a leitura do arquivo de seqüências.

2.5. Considerações Finais

Neste capítulo foi apresentado o contexto deste trabalho, incluindo os resultados de diversos estudos relativos ao BLAST. A seção 2.1 apresenta uma descrição do funcionamento básico do BLAST e suas variações. Uma das conclusões mais importantes obtidas nesta seção é a de que a segunda etapa do algoritmo básico do BLAST, na qual ocorre o maior número de leituras do disco, não possui mecanismos de compartilhamento de memória entre os processos BLAST, tornando-se ineficiente em diversas situações. Entretanto, esta etapa possui duas características importantes:

- O arquivo de seqüências é lido do início ao fim;

– Não é importante a ordem em que é realizada a comparação da seqüência de consulta com cada seqüência do banco, desde que, ao final desta etapa, todas as seqüências do arquivo de seqüências tenham sido lidas.

Essas características foram a base para a concepção, em Lemos et al. (2003), da estratégia de gerenciamento de *buffer* usando anéis.

Na seção 2.2 foi feita uma análise dos diferentes formatos existentes dos bancos de dados lidos pelo BLAST. O conhecimento geral dos formatos dos arquivos foi de extrema importância, pois foi necessário para a implementação da ferramenta e para algumas importantes decisões que possibilitaram o funcionamento da ferramenta com diferentes versões e implementações do BLAST. De modo geral, o banco de dados possui três arquivos: um de seqüências, um de anotações e um de índices, associando as seqüências às suas anotações. O arquivo de índices possui ponteiros para posições dos outros dois arquivos, o que criou algumas dificuldades durante a elaboração deste trabalho, que serão descritas no capítulo 3. Uma importante observação é a de que os arquivos de seqüências de ambos os formatos estudados são muito semelhantes, sendo que o caractere separador de seqüências possui o mesmo código (zero).

Para complementar a análise do algoritmo do BLAST, foi realizada, na seção 2.3, uma análise dos padrões de leitura do banco feitos pelo BLAST. Foi possível observar a leitura do arquivo de seqüências do início ao fim durante a segunda etapa do algoritmo e constatar que nessa etapa há o maior número de leituras do disco. Esse estudo foi essencial também para possibilitar a identificação em tempo de execução da etapa do algoritmo que está sendo executada por um processo BLAST, o que é feito pelo BioProvider, já que o código do BLAST não deve ser modificado para fornecer esta informação diretamente.

Finalmente, foram estudados, na seção 2.4, os diferentes trabalhos realizados para melhorar o desempenho do BLAST e as propostas existentes. Esses estudos foram importantes para direcionar o trabalho e obter idéias de melhorias nas ferramentas da família BLAST. Neste trabalho foram usadas algumas das idéias dos trabalhos apresentados nas seções 2.4.1 e 2.4.2.