FERRAMENTA BIOPROVIDER VOLTADA PARA GRANDES VOLUMES DE DADOS

Aluno: Gabriel Luis Caldeira Vieira
Orientador: Sérgio Lifschitz

Introdução

A comparação de biossequências é uma das tarefas mais importantes atualmente na área de Bioinformática. Entretanto, ainda não existe um Sistema Gerenciador de Banco de Dados (SGBD) específico para essa aplicação, restando apenas o gerenciamento de *buffer* padrão fornecido pelo sistema operacional.

As ferramentas da família BLAST são as mais utilizadas hoje em dia, e o seu desempenho é um fator muito importante, sendo que pequenas melhoras nestas podem trazer grandes benefícios.

Com esse propósito foi criado o BioProvider [1], uma ferramenta provedora de dados que realiza um gerenciamento de *buffer* eficiente para o BLAST. Essa ferramenta é utilizada de forma transparente ao BLAST, já que a comunicação entre ambos é feita através de um *driver* que substitui as chamadas de funções de leitura e escrita no arquivo do banco de dados.

Contudo, na época em que a ferramenta BioProvider foi desenvolvida, as bases de dados disponíveis dificilmente ultrapassavam 1GB, como descrito em Noronha, M. (2007). Ainda que ela tenha sido projetada de forma que o seu desempenho seja melhor para bases grandes, não se imaginava que isso poderia gerar problemas se o tamanho da base ultrapassasse um certo limite.

Numa etapa de pré-processamento da ferramenta BLAST, um programa chamado *formatdb* é chamado para descompactar a base no formato FASTA, separando as sequências da base dos demais dados complementares, como as anotações, gerando ao mesmo tempo um arquivo de índices que une os dois. Nessa etapa, o *formatdb* quebra em pedaços menores os arquivos de índice e de anotações de tal forma que os arquivos sejam mais fáceis de manusear. Isso impede o funcionamento do BioProvider, já que este foi desenvolvido sem tratamento especial para esse caso. Portanto, é necessário realizar modificações no código fonte do BioProvider a fim de adaptá-lo para esse caso em particular, o que consiste em uma tarefa certamente trabalhosa, visto que são necessários conhecimentos bem aprofundados na linguagem de programação C, sistema operacional Linux, *drivers* de dispositivos e o algoritmo BLAST, assim como é imprescindível estudar a fundo o gerenciamento de *buffer* do BioProvider e suas demais funcionalidades.

Objetivos

Modificar o código atual do BioProvider a fim de que este possa tratar eficientemente as múltiplas divisões de uma base de dados volumosa (com tamanho em disco superior a 1GB), sem que ele deixe de funcionar de maneira transparente a ferramenta BLAST.

Além do código fonte, também é necessário modificar o *script* auxiliar que executa o programa. Esse *script* é importante e razoavelmente grande, pois configura diversos parâmetros automaticamente e, principalmente, chama programas auxiliares de pré-processamento (como o *formatdb* do NCBI ou o *format_database* do próprio BioProvider). O funcionamento correto do BioProvider depende essencialmente desse *script*.

Para simplificar os testes realizados durante o desenvolvimento, a fim de que estes levem menos tempo para executar, deve-se limitar a memória RAM manualmente em 256MB ou 512MB, mesmo que a máquina disponha de mais memória física. Como o desempenho do BioProvider depende tanto do tamanho do arquivo de entrada como da memória RAM disponível, é interessante também fazer as simulações variando esses dois parâmetros para analisar o seu efeito sobre o desempenho do programa.

Também seria interessante comparar o desempenho do BioProvider com as versões mais atuais do BLAST, que são muito mais eficientes se comparadas com as versões utilizadas para os testes na época em que o BioProvider foi criado, a fim de descobrir se o uso do BioProvider ainda é capaz de trazer ganhos. Atualmente, por exemplo, existe uma versão chamada *Blast++* que já utiliza as características de processamento paralelo, caso estejam disponíveis, além de outros aperfeiçoamentos no algoritmo.

Metodologia

Para que fique mais claro o problema em questão assim como as soluções encontradas, é necessário entender um pouco melhor o funcionamento do algoritmo BLAST assim como da ferramenta BioProvider antes de apresentar as soluções para o problema das bases grandes dividida em volumes.

I. Características do BLAST

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. As anotações das seqüências incluem informações sobre as mesmas, como seus números identificadores e funcionalidades principais. São usadas ferramentas – no caso, *formatdb* – 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.

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.

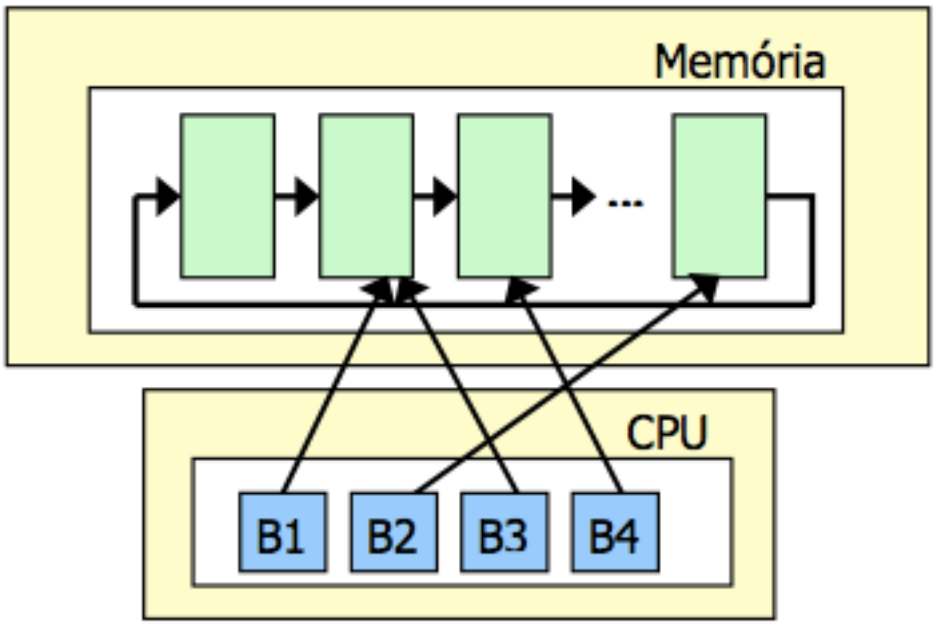
A segunda etapa do algoritmo BLAST possui também uma característica importante: como o objetivo é analisar todas as seqüências do banco de dados a procura de similaridades, não é importante a ordem em que é realizada a comparação da seqüência de consulta com cada seqüência do banco. Logo, um processo 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.

II. A ferramenta BioProvider

Devido às características do BLAST descritas na seção I, 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 abaixo 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.



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.

III. Tratamento para uma base dividida em volumes

Para tratar a possível quebra da base em "volumes" (nome atribuído às partições pela NCBI como pode ser lido no seu próprio site [2]) foram consideradas várias soluções alternativas. Porém, apenas uma permite o BioProvider de continuar operando transparentemente ao BLAST.

A primeira solução analisada constava em modificar superficialmente o programa *formatdb*, afim de que ele não quebrasse mais as bases quando elas fossem muito volumosas. Embora não haja muita documentação sobre esse comportamento particular da ferramenta *formatdb*, segundo a NCBI, numa página que explica os parâmetros do formatdb e fastacmd [2], uma base tem será particionada obrigatoriamente caso ela seja muito grande, isto é, se esse procedimento não fosse importante haveria um parâmetro para desativá-lo no próprio formatdb e, além disso, ela explica que uma base quebrada em pedaços menores é mais fácil de manusear – entende-se que há um ganho de desempenho ao fazer isso. Em outras palavras, essa modificação possivelmente faria com que o BLAST não funcionasse mais ou o tornaria mais lento, além de ser intrusiva de certa forma. Portanto, foi descartada.

Uma segunda solução seria reconsiderar a primeira juntamente com uma modificação no código fonte da ferramenta BLAST afim de garantir que esse funcionaria corretamente com uma base grande não particionada. Contudo, ao fazer isto, todo o esforço despendido anteriormente para tornar o BioProvider transparente ao BLAST seria jogado fora. Além disso, a ferramenta da NCBI é atualizada de tempos em tempos e o seu código fonte não é muito bem documentado.

Finalmente, a solução escolhida involve alterar a maneira como o BioProvider funciona, isto é, modificar o seu código fonte, adicionando um tratamento especial para o caso das bases grandes. Para isso foi necessário estudá-lo a fundo, afim de identificar os pontos onde seriam necessárias alterações. Uma forma de fazer isto é tomando um exemplo e analisando os casos que resultariam em erro decorrentes da lógica de programação incorreta.

Seja uma base dividida em 4 blocos mas grande o suficiente para ser quebrada em 3 volumes pelo *formatdb*, conforme ilustrada pela figura 1. A etapa de quebra em volumes ocorre primeiro, ou seja, para cada um existiriam 4 permutações dos arquivos de índice e anotações. Embora não sejam criadas permutações do arquivo de sequências, ele também é quebrado em volumes.

O primeiro problema aparente é quando o *buffer* do anel chega no final de um volume do arquivo de sequências. O segundo ocorre quando um processo BLAST pedir um arquivo de índices ou de anotações. O BioProvider saberá apenas a permutação correta desse arquivo, mas não o volume.

No caso do primeiro problema, o *buffer* do anel deverá ser reposicionado para o início do próximo volume ou para o início do primeiro volume (isto é, se já tiver percorrido todos), de tal maneira que ele continue percorrendo o arquivo de sequências ciclicamente. O BioProvider deverá ter controle do volume corrente que está sendo percorrido pelo buffer.

No segundo caso, dependendo da implementação do BioProvider, poderia ser que não fosse necessária nenhuma alteração no código fonte, caso ele devolvesse o nome dos arquivos pedidos apenas concatenando o sufixo da permutação correta (por exemplo, se o BLAST pedir o arquivo "nr.00.phr" e o BioProvider identificar a permutação desse bloco como sendo aquela de índice 2, basta que ele apenas concatene "_2" antes do ".phr" para devolver o arquivo correto "nr.00_2.phr"). Infelizmente ele não o faz dessa forma, logo é necessária uma alteração no código um pouco mais complicada através de uma manipulação de *ponteiros*.

Alteração realizadas no código fonte

Embora não tenha sido possível terminar todas as alterações necessárias para testar uma nova versão do BioProvider, uma parte (a grande maioria das características da ferramenta vai seguir inalterada) do código fonte foi estudada e modificações foram feitas. No entanto, foram encontradas algumas dificuldades durante a programação.

Primeiramente, as modificações devem ser feitas de forma a alterar o código original quanto menos possível, visto que não é nada trivial testar ou realizar um *debug* no BioProvider várias vezes seguidas – como se faz normalmente ao escrever um programa de menor porte – pois o script leva vários minutos para preparar a base de dados antes de executá-lo. Ou seja, é preciso escrever o código atenciosamente porque se um erro for introduzido, poderá ser muito complicado e custoso em termos de tempo de removê-lo depois.

Em seguida, afim de que outros estudantes e pesquisadores possam trabalhar sobre o código já modificado, é interessante criar uma documentação, por menor que seja, das partes alteradas. Portanto, todas as alterações no código foram descritas em um arquivo texto contendo a linha e descrição da modificação feita. Além disso, esse mesmo arquivo também contém um novo formato para o arquivo de configuração do BioProvider (*config*), já que foi necessário inserir alguns parâmetros que antes não eram necessários, como o nome do arquivo *alias* ou então os nomes de todos os volumes de cada bloco.

Próximas alterações

Além de dar continuidade a programação descrita no arquivo texto, seria interessante modificar o código do BioProvider e do *script* auxiliar afim de minimizar o tempo necessário para preparar a ferramenta antes de cada teste, mesmo que algumas funcionalidades fossem desligadas. Para diminuir o tempo entre cada teste ainda mais, uma base de dados artificialmente truncada para ser pequena – isto é, o tamanho no limite que o *formatdb* começa a quebrar a base em volumes – poderia ser usada no lugar de uma base de verdade.

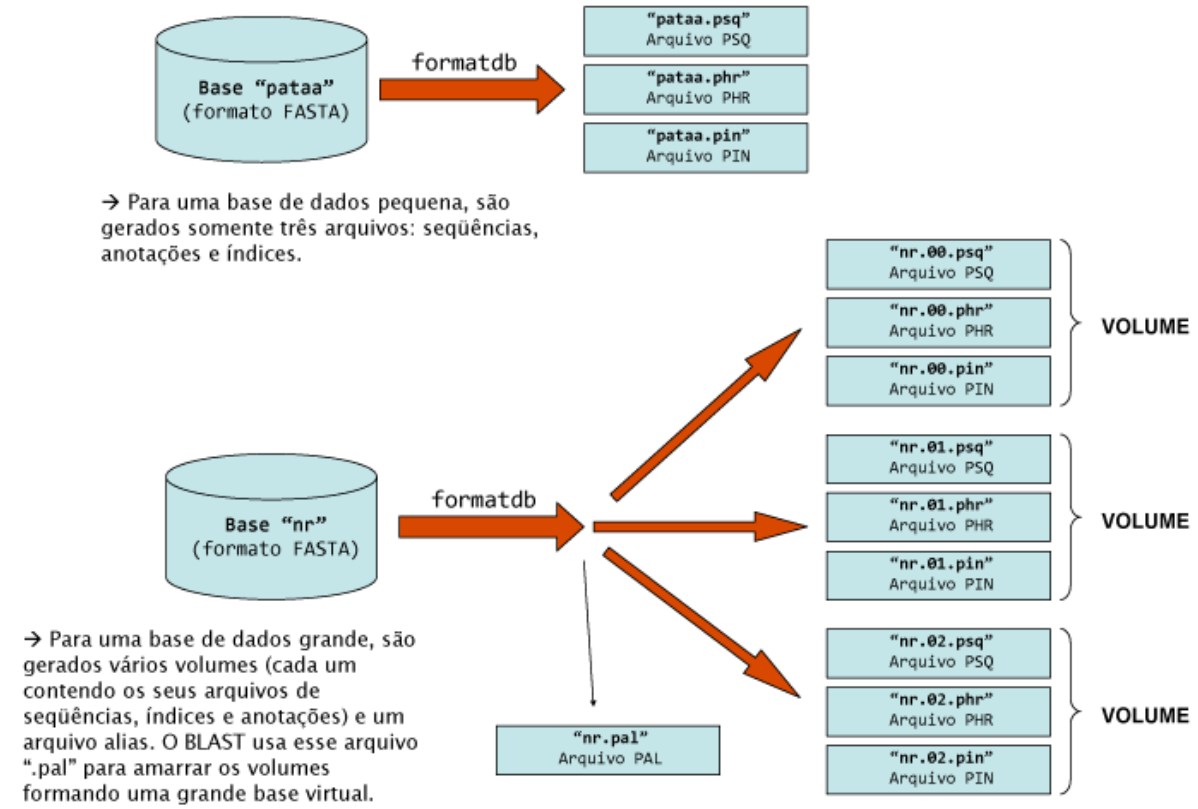
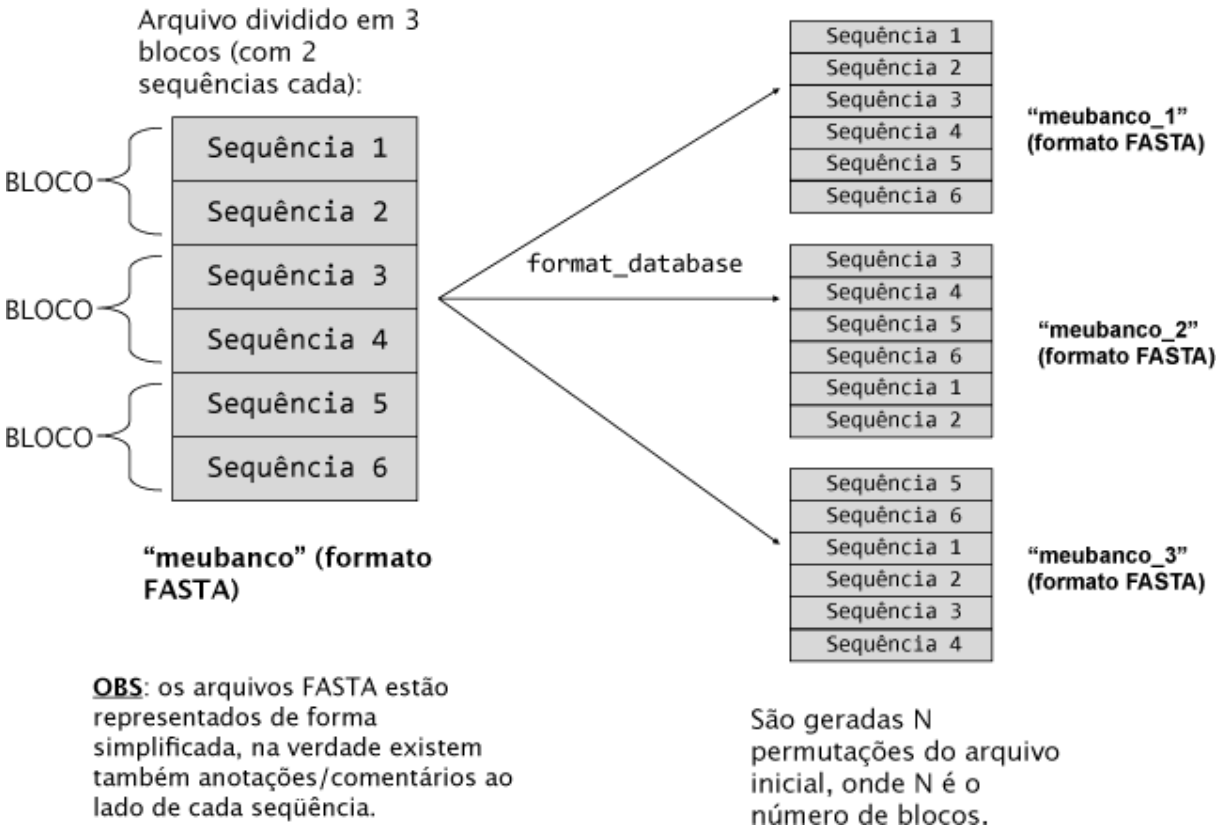


Figura 1 (acima): esquema ilustrando uma base suficientemente grade para ser quebrada em 3 volumes.

Figura 2 (abaixo): esquema ilustrando o funcionamento do programa *format_database*.



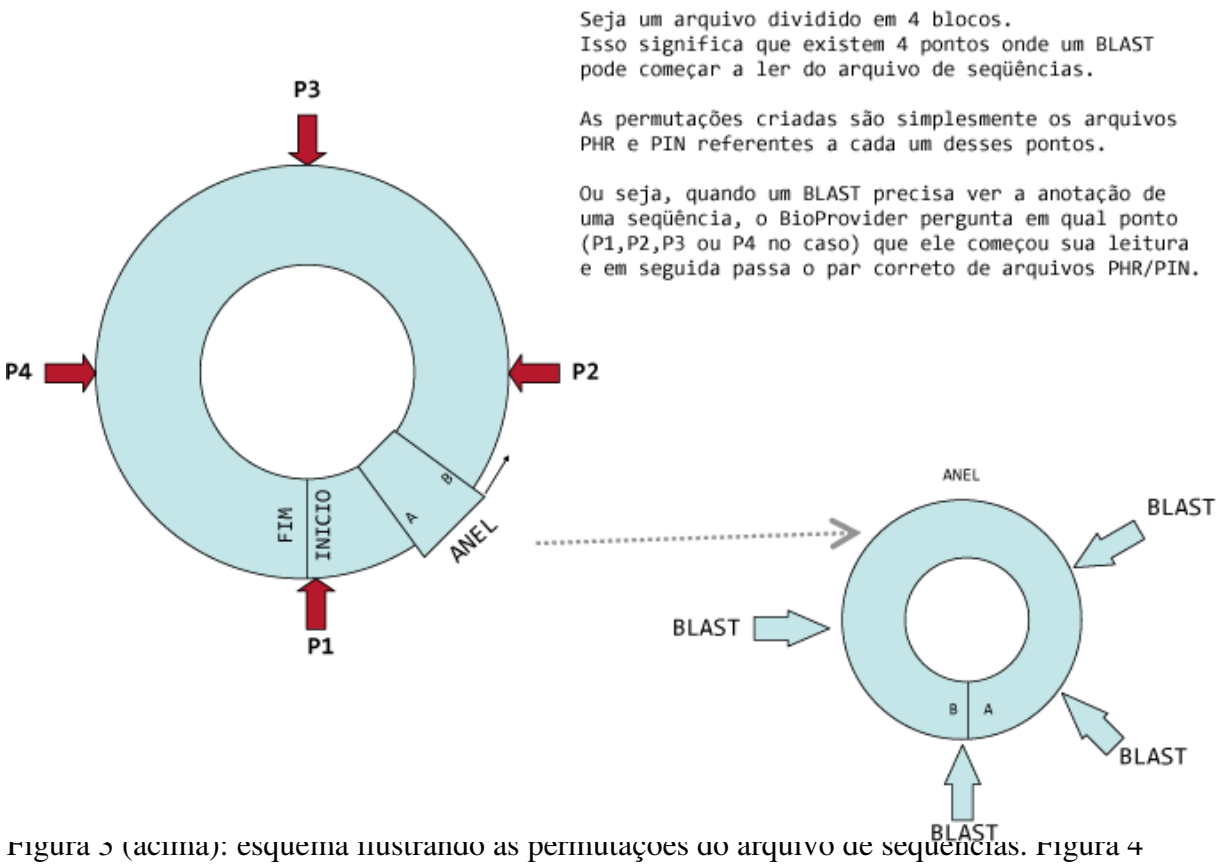
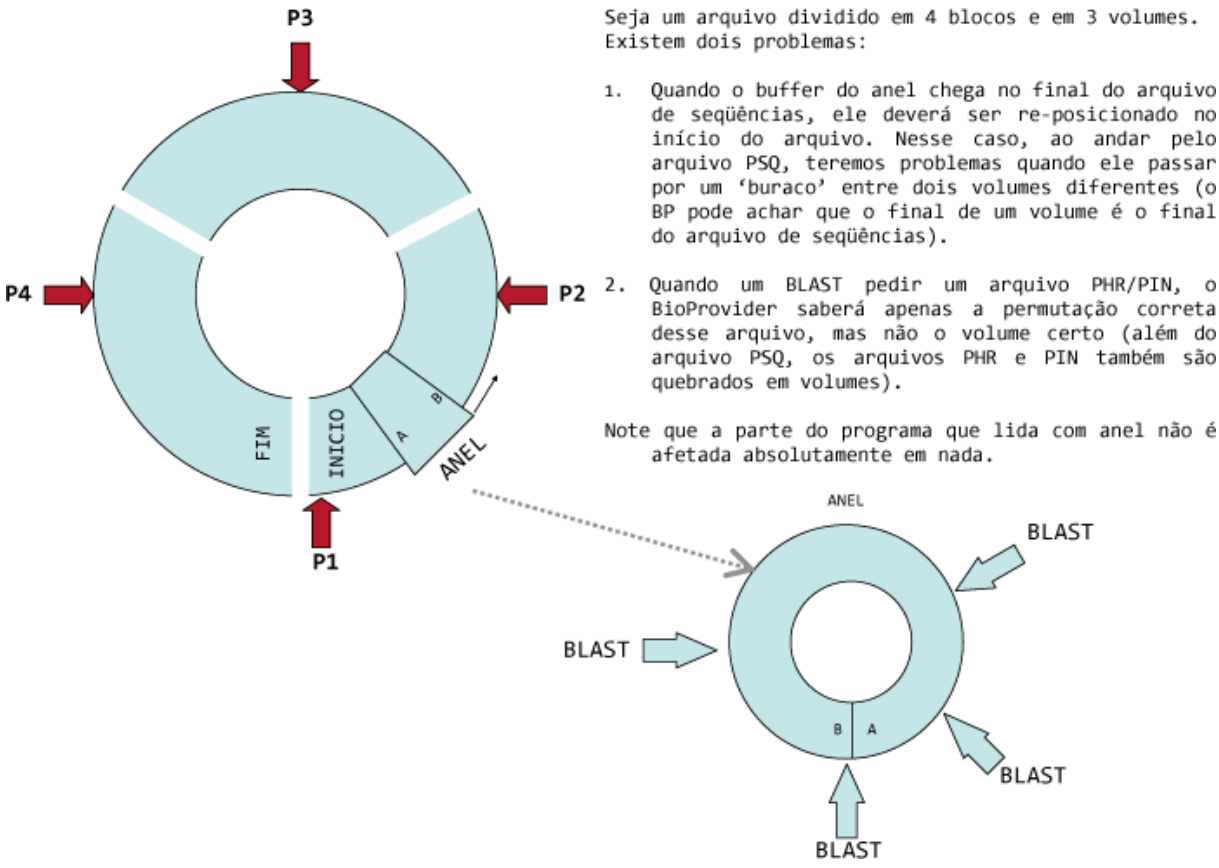


Figura 3 (acima): esquema ilustrando as permutações do arquivo de seqüências. Figura 4 (abaixo): esquema ilustrando a mesma situação para uma base volumosa.



Conclusão

De fato, a ferramenta funciona como o esperado para tamanhos inferiores a esse limite de 2GB, mas atualmente os pesquisadores dificilmente trabalham com bases que não o ultrapassem. Logo, antes que a ferramenta seja publicada como projeto de código aberto (na verdade, já existe uma versão estável para bases pequenas disponível para download no site do Laboratório de BioInformática da PUC-Rio), seria interessante finalizar as alterações no código que permitirão ao BioProvider tratar as múltiplas partições (ou "volumes") de uma base de dados de tamanho superior a 1GB.

Pode-se concluir que a realização desse projeto consiste em uma tarefa bastante complexa, visto que são necessários conhecimentos bem aprofundados na linguagem de programação C, sistema operacional Linux, drivers e do algoritmo BLAST, além de um estudo a fundo da ferramenta BioProvider.

Contudo, essa complexidade é justificável, pois trata-se de um projeto inovador que pode trazer muitos ganhos para a comunidade científica, considerando a utilização frequente das ferramentas da família BLAST na área da Bioinformática.

Referências

1 - NORONHA, M.; **Controle da Execução e Disponibilização de Dados para Aplicativos sobre Sequências Biológicas: o Caso BLAST**. Rio de Janeiro, 2007. 83p. Dissertação de Mestrado, Departamento de Informática, PUC-Rio.

2 - http://www.ncbi.nlm.nih.gov/staff/tao/URLAPI/formatdb_fastacmd.html