Universidade Federal do Rio Grande do Sul

Instituto de Física

Bacharelado em Física

Inferência de raiz evolutiva de genes como base para a simulação de crescimento de genomas

Aluno: Glaucio Teixeira Souza

Orientadora: Rita Maria Cunha de Almeida

Monografia realizada sob a orientação da professora Rita Maria Cunha de Almeida, apresentada ao Instituto de Física da UFRGS em preenchimento parcial dos requisitos para obtenção do título de Bacharel em Física.

Porto Alegre, Dezembro de 2010

Abstract

Genome can be defined as any hereditary information of an organism and this information is encoded in DNA. Knowing the nucleotide sequence of a DNA molecule, we can identify their genes and study them. With advances in genetics over the past decades it has been possible to decode and map the genome of various species and compare them what allow us to investigate how these species are related and find what evolutionary paths that led to the diversity of organisms observed today. Understanding the evolution of the genome is fundamental to understanding the evolution of organisms. As a basis for studying the evolution of genomes we used the approach of gain/penalty, Mirkin et al. [1], for the inference of ancestral genes common to orthologous groups of genes present in modern organisms.

Resumo

O genoma pode ser definido como toda a informação hereditária de um organismo e esta informação está codificada no DNA. Conhecendo a sequência de nucleotídeos de uma molécula de DNA poderemos identificar os genes que ela contém e assim estudá-los. Com os avanços da genética ao longo das últimas décadas foi possível decodificar e mapear o genoma de várias espécies e com isso comparar suas sequências de genes o que permite pesquisar como tais espécies estão relacionadas, isto é, quais os caminhos evolutivos que levaram à diversidade de organismos observadas atualmente. Compreender a evolução do genoma é fundamental para compreender a evolução dos organismos. Como base para o estudo de evolução de genomas utilizamos a abordagem de ganho/penalidade, Mirkin et al. [1], para a inferência dos genes ancestrais comuns a grupos de genes ortólogos presentes nos organismos atuais.

Sumário

Capítulo 1

Introdução

Cada vez que uma célula se divide, todas as moléculas de DNA devem ser copiadas para as células resultantes, de modo que estas possuam uma cópia completa do genoma e com isso ter a capacidade de decodificar as informações contidas nos genes, se desenvolver e desempenhar suas funções específicas. No processo de cópia, mutações podem ocorrer com bastante frequência, mas também pode haver dano devido à insultos ao DNA ocasionados por radicais livres, por radiação ultravioleta, etc. Essas alterações podem resultar também em mutações, as quais, ao longo do tempo, serão a base para a evolução do genoma. Outra fonte importante de evolução do genoma são duplicações de genes pois permitem que genes muito especializados possam evoluir sem prejudicar o funcionamento do organismo. Estima-se, inclusive, que alguns organismos tenham duplicado o genoma inteiro algumas vezes.

Para representar a história evolucionária inferida para um grupo de organismos, a qual chamamos filogenia, utilizamos um diagrama chamado árvore

filogenética. Essa representação permite que visualizemos como as espécies estão relacionadas entre si e como evoluíram ao longo do tempo. Por exemplo, podemos observar praticamente todo o passado evolutivo do ser humano, desde os organismos mais simples até o ponto em que nossos ancestrais se separaram de outros primatas e evoluíram até o homo sapiens.

Neste trabalho iremos estudar grupos ortólogos, os quais podem ser definidos como conjuntos de genes ortólogos, isto é, genes de diferentes espécies, mas que possuam um gene ancestral comum. Construindo uma árvore filogenética para um grupo ortólogo podemos inferir como se deu a evolução para um determinado gene do grupo. Esses dados são essenciais para guiar ou verificar hipóteses para dinâmica de crescimento de genomas.

Este trabalho tem como objetivo sistematizar a determinação da raiz ancestral, o gene ancestral comum mais recente, para um gene individual presente em um dado grupo de genes ortólogos, a partir da análise da árvore filogenética como formulado por Mirkin [1], isto é, buscando o cenário evolutivo mais parcimonioso para um grupo de ortólogos considerado. Esta sistematização tomará a forma de uma ferramenta computacional para a inferência da raiz evolutiva de todos os genes de um genoma a partir da rede filogenética de consenso, isto é, uma árvore filogenética atualmente considerada válida pelos biólogos.

Capítulo 2

Definições

2.1 Mutações e a evolução do genoma

O genoma ao evoluir pode tanto adquirir novos genes quanto perder genes existentes. A forma como são adquiridos parece determinar muitas das características das redes genômicas. Essa aquisição se dá, basicamente, por duplicação de genes, de cromossomos ou de todo o genoma, e pela formação de novos genes.

Durante o processo de duplicação de genes podem ocorrer mutações, alterações nas sequências de DNA, as quais podem modificar genes existente e até mesmo formar novos. Considerável parte destas alterações são neutras, nas quais o DNA foi modificado de tal forma que a informação armazenada não é afetada e as proteínas são decodificadas corretamente. Consequentemente as células resultantes seguem realizando suas funções normalmente.

Entretanto podem ocorrer modificações danosas, que prejudiquem o funcionamento adequado da célula, ou benignas, as quais eventualmente podem propiciar alguma vantagem ao organismo.

De acordo com Koonin [2], os eventos elementares para evolução de genes podem ser classificados como:

(i) transferência vertical com modificações: quando o gene é herdado por espécies resultantes de especiação e após terem sofrido modificações(mutações);

(ii) duplicação de genes, seguida de transferência vertical com modificações: caso em que genes são duplicados em um organismo, mutam e depois esses genes são herdados pelas gerações futuras;

(iii) perda de gene: situação em que um gene é perdido após a especiação ;

(iv) transferência horizontal: transferência de material genético entre espécies;

(v) fusão, fissão e outros reordenamentos dos genes: eventos em que pode haver fusão, fissão ou reordenamento dos nucleotídeos que compõem o DNA, que por sua vez formam os genes, que ao serem modificados levam a ocorrência desses eventos.

2.2 Genes ortólogos

Charles Darwin propôs que "todas as formas de vida orgânica que já viveram na Terra descendem de algum organismo primordial", ou seja, que os organismos atuais descendem de outros mais ancestrais. Partindo dessa premissa, podemos supor que tais organismos possuam material genético em comum,

isto é, genes com trechos de seu DNA em comum. Estes organismos são classificados como homólogos.

Homologia, em uma definição mais abrangente, representa um relacionamento entre quaisquer entidades que possuam uma origem comum. Os genes relacionados por homologia, isto é, que compartilham uma origem comum, são ditos homólogos.

Aperfeiçoando esta classificação, introduzimos os termos genes ortólogos e parálogos. Genes parálogos estão relacionados entre si pela duplicação, ocupando duas posições diferentes no mesmo genoma. Parálogos geralmente possuem funções idênticas ou similares. Os genes ortólogos, objeto de estudo neste trabalho, são aqueles relacionados entre si através da especiação, quando uma espécie evolui em duas espécies distintas. Os ortólogos são similares entre si devido à sua origem comum e geralmente, mas nem sempre, possuem a mesma função. O estudo da diferença genética entre ortólogos, isto é, a análise da similaridade entre os trechos de DNA que os compõem, pode ser utilizado para inferir o relacionamento entre organismos.

Os dados necessários para analisar a ortologia dos genes em bancos de dados como COG (Cluster of Orthologous Groups) que contém grupos de ortólogos para organismos eucariotos (KOGs) e grupos de ortólogos para organismos procariotos (COGs). Estes dados podem ser obtidos de base de dados como o STRING (Search Tool for the Retrieval of Interacting Gene/Proteins) [4]. O STRING é um banco de dados e uma ferramenta web dedicada a interações proteína-proteína incluindo interações físicas e funcionais. A infraestrutura básica do STRING inclui um grupo de genomas completamente seqüenciados e classificações de ortologia, combinando informações de diversas fontes incluindo repositórios experimentais, métodos

computacionais de predição e coleções de textos públicos e assim, atua como um meta-banco de dados que mapeia todas as evidências de interação em um conjunto de genomas e proteínas.

2.3 Árvore Filogenética

Podemos dizer que a evolução das espécies é um processo de ramificação. Com o passar do tempo espécies tendem a se dividir, dando origem a duas novas espécies e esses novos ramos evoluem independentemente, sofrendo novas modificações e eventualmente dando origem a outras espécies. Ou seja, evolução leva à especiação e ao longo do tempo o somatório desses eventos gerou o que alguns biólogos chamam de árvore da vida.

Uma árvore da vida ou árvore filogenética, na nomenclatura científica, é um diagrama geralmente utilizado para representar a história das espécies, permitindo visualizar todas as ramificações decorrentes dos eventos de especiação.

A filogenética estuda o parentesco evolutivo entre organismos, os quais podem ser determinados com base em suas características morfológicas e adicionalmente, com o advento do sequenciamento do genoma de diversos organismos, utilizamos dados moleculares considerando que organismos próximos, em relação à sua evolução, possuem semelhanças em suas estruturas moleculares de DNA e proteínas. Com base nesses dados podemos aperfeiçoar a determinação da topologia de árvores para grupos de organismos inferindo árvores filogenéticas que descrevam caminhos evolutivos mais prováveis. Esta árvore, aceita como a que possui maior probabilidade de ser a

representação da evolução dos organismos considerados, é chamada árvore de consenso.

Neste momento se faz útil definirmos a nomenclatura utilizada ao nos referirmos à árvore e seus componentes.

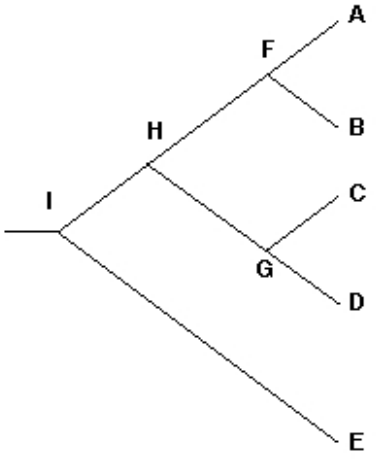

Figura 2.1: Elementos da árvore

A Figura (2.1) apresenta os elementos que compõem a árvore. Os elementos identificados pelas letras de A até E são denominados nós externos ou nós folha, e podem representar espécies, proteínas, genes, etc., existentes. As letras F até I identificam os nós internos, os quais representam nós ancestrais, ou seja, espécies não mais presentes nos dias de hoje. Definimos um nó pai como o ancestral imediato de um dado nó, o qual chamamos de nó filho. Na Figura (2.1) F é nó pai dos nós filhos A e B, enquanto o nó ancestral I, que é a raiz da árvore pois é o último ancestral comum dos organismos existentes, é nó pai dos organismos representados por H e E. Também podemos definir como topologia da árvore a ordem dos nós na mesma. A árvore da

Figura (2.1) possui 4 níveis, partindo da raiz para os nós externos. A raiz está no primeiro nível, seguida do nível do nó H, o terceiro nível contendo os nós F e G e o último com os nós A,B,C,D e E.

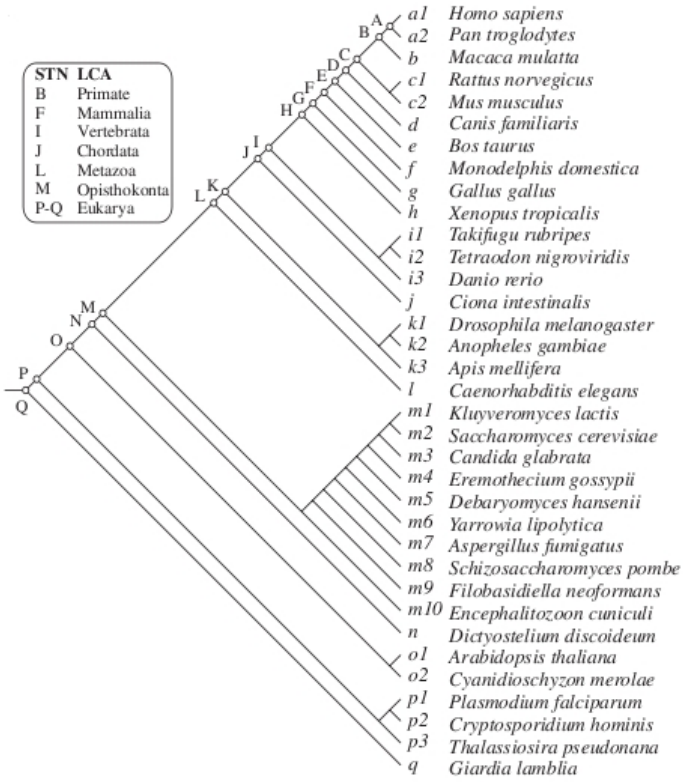

Figura 2.2: Árvore filogenética de consenso.

Dado que a construção de uma árvore não é o foco deste trabalho, vamos utilizar uma árvore de consenso apresentada na Figura (2.2) baseada na integração manual de uma variedade de filogenias [6-11]. Esta árvore foi construída abrangendo os 35 eucariotos com genomas completamente seqüenciados na época em que o artigo [3] foi publicado. A árvore da Figura (2.2) foi ordenada de modo que o primeiro organismo fosse o Homo sapiens, mas não precisa, necessariamente, ser dessa forma. O primeiro organismo poderia ser Macaca mulatta, por exemplo. Todos os organismos que estão vivos hoje evoluíram durante o mesmo tempo desde o início da vida, de modo que

qualquer um pode ser colocado no topo da árvore, respeitando a hierarquia de nós pai-filho, isto é, as bifurcações permanecem as mesmas, mas a forma como são ordenadas na árvore pode ser diferente. Por exemplo, na Figura (2.2) os nós a1 e a2 poderiam ser invertidos, de modo que a primeira espécie na árvore, de cima para baixo, seria *Pan troglodytes* e a segunda passaria a ser *Homo sapiens*.

Na figura (2.2), cada nó nomeado com uma letra maiúscula (A-Q), representa o ancestral comum dos organismos existentes, e o nó Q é o último ancestral comum para todos os organismos representados.

Estudando genes pertencentes a genomas de diferentes espécies é possível, a princípio, reconstruir o cenário evolutivo de cada gene. Construindo uma árvore filogenética para um grupo ortólogo, podemos inferir como se deu a evolução para um determinado gene do grupo.

Capítulo 3

Desenvolvimento

Para investigar como se dá a evolução de genes, o Grupo de Modelos Teóricos e Computacionais têm feito medidas experimentais da plasticidade, medida do quanto uma rede de genes é tolerante a modificações nos seus genes. Uma rede muito plástica pode sofrer muitas modificações, já redes pouco plásticas não toleram mutações. Plasticidade baixa está associada a redes genéticas muito conservadas, que podem estar muito otimizadas, não tolerando mudanças as quais poderiam facilmente prejudicar o desempenho de suas funções ou até mesmo inviabilizá-las. Redes pouco plásticas presentes em organismos que não sofreram modificações sugerem redes primitivas, enquanto que se ainda estão sendo modificadas podem ser recentes e estarão presentes em poucos organismos.

As medidas de plasticidade são realizadas pela análise da abundância e da diversidade dos KOGs [3]. O conceito de abundância reflete a quantidade de ortólogos existentes por organismos e a diversidade apresenta como é a

distribuição de ortólogos nos organismos, de um dado grupo ortólogo.

No artigo apresentado por Castro et al. [3] o estudo é focado na origem das redes de genes humanos da apoptose e de estabilidade genômica, então é natural buscar por genes ortólogos a esses para poder determinar seus ancestrais comuns.

Tendo determinado um gene ou conjunto de genes a ser estudado, verificamos a qual grupo ortólogo ele pertence e com essa informação obtemos todas as espécies que possuem genes ortólogos ao gene objeto de estudo. Todas essas informações podem ser obtidas a partir da base de dados do STRING.

3.1 Abordagem

A partir do grupo de genes ortólogos ao gene escolhido, podemos, utilizando os dados obtidos do STRING, determinar quais espécies possuem algum desses genes e a partir daí mapear o padrão filético de genes ortólogos, isto é, o padrão de presença/ausência do gene nas espécies consideradas.

Segundo Mirkin et al. [1], a reconstrução do cenário evolutivo para um conjunto individual de genes ortólogos pode ser determinada a partir de uma dada árvore de espécies e um grupo de ortólogos com seu particular padrão filético. Utilizando estas informações devemos encontrar o mapeamento mais parcimonioso para o conjunto de ortólogos formado pelos nós internos da árvore de modo que resultem no padrão filético observado para o grupo de ortólogos escolhido. O cenário evolutivo mais parcimonioso corresponde ao

cenário que apresente o menor número possível de eventos evolutivos, como os citados na seção 2.1, para uma dada árvore filogenética.

Mas, para construirmos os cenários evolutivos com base nos dados de ortologia, precisaremos realizar uma considerável simplificação: iremos considerar somente dois eventos, o surgimento do gene, independente da forma como ocorra, e a perda do gene. Esta aproximação se faz necessária pois não há informação suficiente para distinguir, nos organismos ancestrais, qual evento foi responsável pelo quê. Sendo assim, o cenário evolutivo mais parcimonioso será aquele que combine a menor quantidade de eventos de surgimento e perda de genes nos organismos ancestrais de modo a obter o padrão filético observado atualmente. Ao analisar a árvore, identificamos o relacionamento entre um nó pai e um nó filho para determinar o surgimento (ou perda) de um gene. Um surgimento terá ocorrido quando o gene não estiver presente no nó pai mas estiver presente no nó filho, enquanto que se o gene estiver presente no nó pai, mas não estiver no nó filho, houve uma perda de gene.

A análise de parcimônia utilizando a abordagem de ganho/penalidade pode ser sintetizada em uma equação de custo:

$$S = \lambda + \gamma \tag{3.1}$$

onde a variável λ representa o número de eventos de perda do gene, γ o número de eventos de surgimento do gene e o custo S é denominado valor de inconsistência para o KOG. Assim o cenário mais parcimonioso será aquele que minimizar a inconsistência para o KOG considerado. Para exemplificar esta abordagem, vamos analisar os exemplos a seguir.

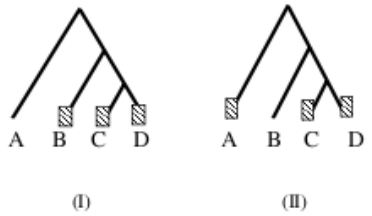

Figura 3.1: Exemplo de padrão filético de presença/ausência de genes para dois grupos de ortólogos distintos (I) e (II) para as espécies atuais A, B, C e D. Retirado de Mirkin et al.[1]

Na Figura 3.1 temos a representação de uma árvore filogenética contendo quatro espécies atuais: *A, B, C* e *D*. A situação (I) representa o padrão filético para um dado grupo de genes ortólogos e a (II) o padrão para um outro grupo. Vamos analisar cada um dos casos em separado.

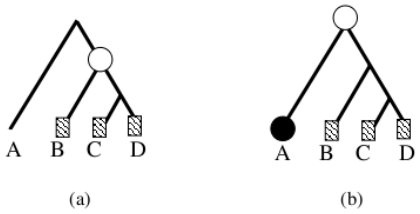

Figura 3.2: Padrão filético de presença/ausência de genes para o caso (I). Retirado de Mirkin et al.[1]

Na Figura (3.2) temos duas possibilidades: Na situação da Figura (3.2a) o gene surgiu no ancestral comum das espécies *B,C* e *D* e na situação da Figura (3.2b) o gene surgiu no ancestral comum a todas as espécies e perdeu-se posteriormente em *A*. Considerando a equação (3.1) o cenário mais parcimonioso é o da Figura (3.2a), pois há somente um evento, um surgimento do gene que foi herdado até as espécies *B,C* e *D*, de modo que o nó intermediário também possui um gene ortólogo. Para esse caso $S = 1$. Na situação da Figura (3.2b) dois eventos ocorreram, um surgimento e uma perda, assim $S = 2$. Para esse caso, os outros nós intermediários também possuem o gene, de modo que esse seja herdado até chegar às espécies *B,C* e *D*. Qual-

quer outra combinação de surgimento/perda de gene irá levar a um valor de inconsistência maior do que $S = 1$.

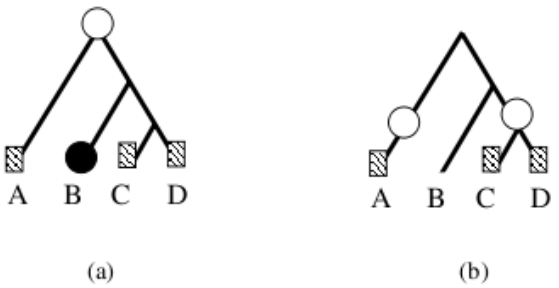

Figura 3.3: Padrão filético de presença/ausência de genes para o caso (II). Retirado de Mirkin et al.[1]

Na Figura (3.3) o padrão filético é ligeiramente diferente, mas aborda uma situação mais interessante. Temos em (a) dois eventos, um surgimento na raiz ancestral da árvore e uma perda em B, o que resulta em $S = 2$. No caso (b), também temos dois eventos, mas agora ambos são surgimentos. Um surgimento em A e outro no ancestral comum de C e D. Neste caso também teremos $S = 2$. Surge uma degenerescência.

Para acabar com essa degenerescência, nos baseamos em considerações biológicas as quais suportam que, em média, a perda de genes pode ser um evento evolutivo muito mais provável do que um surgimento. Quando os genes evoluem e se modificam, como já foi discutido na seção 2.1, tais alterações podem ser devidas a uma variedade de eventos. É muito mais provável que uma modificação leve à perda de um gene do que ao surgimento, pois ao longo do tempo, modificações levam à sequências de nucleotídeos tão diferentes o que torna muito improvável que ocorra um ordenamento de forma a propiciar o surgimento de um novo gene ortólogo. Logo a Figura (3.3a) apresenta a situação mais parcimoniosa. Para modelar essa consideração precisamos modificar a equação (3.1) introduzindo um peso g para o caso de

surgimento de gene, o que resulta em:

$$S = \lambda + g\gamma \tag{3.2}$$

Assumindo o peso $g = 2$ na equação (3.2), como utilizado na literatura [1, 3, 12, 13], teremos o caso mais simples que diferencia o surgimento (mais improvável) da perda (mais provável) para os genes ortólogos. Com isso, a Figura (3.3a) resulta no cenário evolutivo mais parcimonioso para o padrão filético (II), apresentando um valor de inconsistência $S = 3$, enquanto no outro caso da Figura (3.3b) teremos $S = 4$.

3.2 Algoritmo

Modelar a abordagem utilizada na seção 3.1 não é nada trivial. Será preciso a utilização de uma estrutura de dado mais complexa, mais especificamente o uso da estrutura de árvores binárias.

Árvore binária é uma estrutura em árvore para armazenamento de dados a qual permite a representação hierárquica dos dados. Uma árvore é composta por nós, estruturas que armazenam os dados, que possuem relacionamento hierárquico entre si. O primeiro nó da árvore é chamado de raiz e os nós subsequentes são seus nós filhos. Um nó que possua pelo menos um filho é chamado de nó pai. Para a modelagem da árvore filogenética utilizaremos uma árvore binária, uma árvore na qual cada nó possui no máximo dois nós filhos.

Após a modelagem da árvore de consenso, um vetor contendo o padrão

filético para o KOG considerado é lido e armazenado nas variáveis correspondentes aos nós externos. A seguir a árvore começa a ser analisada por partes. Nesta primeira implementação do programa foram criados três blocos (sub-árvores) para os quais são analisadas todas as configurações possíveis de surgimento e perda de genes e calculado para cada uma o valor de inconsistência. O menor valor determinará a configuração a ser fixada e então, após terem sido definidos os padrões de presença/ausência para todos os nós internos das sub-árvores, o programa calcula o restante dos nós internos, repetindo o mesmo processo de análise. Após ter sido calculado o menor valor de inconsistência, este será apresentado juntamente com o índice do nó da árvore que representa o último ancestral comum ao grupo de ortólogos. Também é exposta uma representação simplificada da árvore contendo o padrão de presença/ausência dos genes.

A ideia de utilizar sub-árvores foi considerada devido à quantidade de configurações existentes a serem analisadas. Considerando que a árvore de consenso possui 34 nós internos, serão 2^{34}, ou seja, $\sim 1,17\text{x}10^{10}$ combinações possíveis. Passando a utilizar blocos de nós internos, o número de combinações a serem analisadas passou para 1.321.000.

Para determinar o cenário evolutivo mais parcimonioso ao percorrer todas as configurações possíveis do padrão de presença/ausência dos genes, buscamos o padrão que minimize a equação:

$$S = V_{raiz} + \sum_{N} \gamma V_i (V_{i+1} - V_i)^2 + V_{i+1}(V_{i+1} - V_i)^2 \qquad (3.3)$$

onde cada V pode assumir o valor de presença (1) ou de ausência (0) do gene para o organismo em questão e γ é o peso de surgimento do gene. O valor de inconsistência, S, é dado pela soma do valor do nó raiz (V_{raiz}) mais

o somatório sobre N, o número de nós da árvore, isto é, o número total de organismos desde os mais ancestrais até os atuais, contemplados na árvore filogenética de consenso. V_i é o valor do nó considerado no passo atual e V_{i+1} o valor de seu nó ancestral imediato.

A Figura 3.4 apresenta o diagrama de atividades representando o fluxo de execução do algoritmo e no Apêndice A está disponível o código fonte do programa.

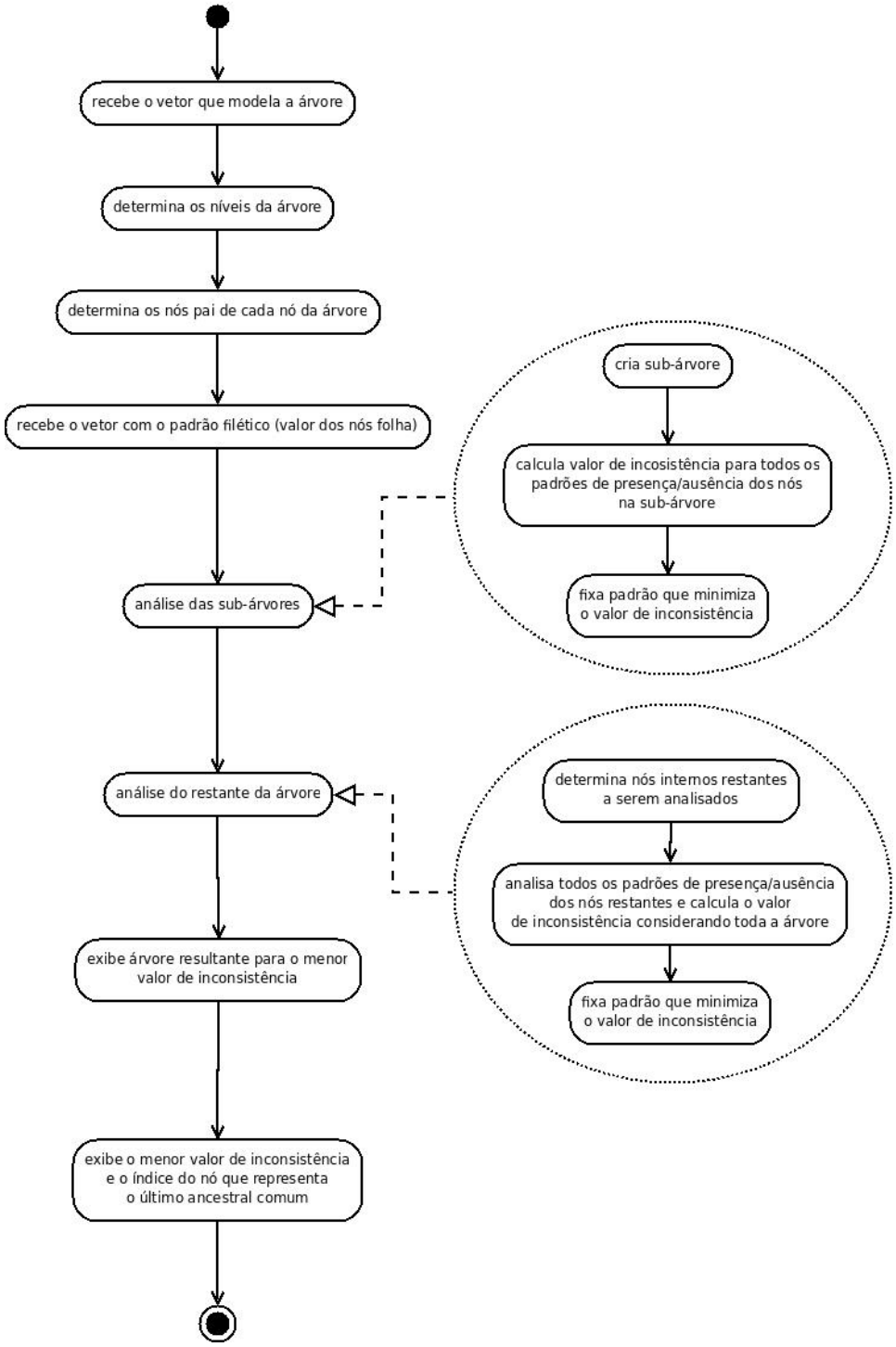

Figura 3.4: Fluxo de execução do algoritmo

Capítulo 4

Resultados

O programa permitiu a reprodução das inferências dos cenários evolutivos dos KOGs utilizados no artigo de Castro et al. [3] os quais foram inferidos utilizando a mesma abordagem discutida na seção 3.1 mas calculados manualmente. Alguns dos resultados obtidos serão apresentados nessa seção.

A representação da árvore filogenética dos genes para o gene LIG3 presente no KOG4437 é apresentada na Figura (4.1). O valor de inconsistência encontrado pelo programa, Figura (4.1b), $S = 5$, é o mesmo que foi encontrado via cálculo manual em [3], Figura (4.1a), assim como o padrão de presença/ausência. O último ancestral comum, o primeiro surgimento do gene, está no nó circulado em vermelho. Os eventos de perda de gene estão circulados em preto.

Na Figura (4.2) temos a representação da árvore filogenética contendo o padrão de presença/ausência dos genes para o gene MSH5 presente no

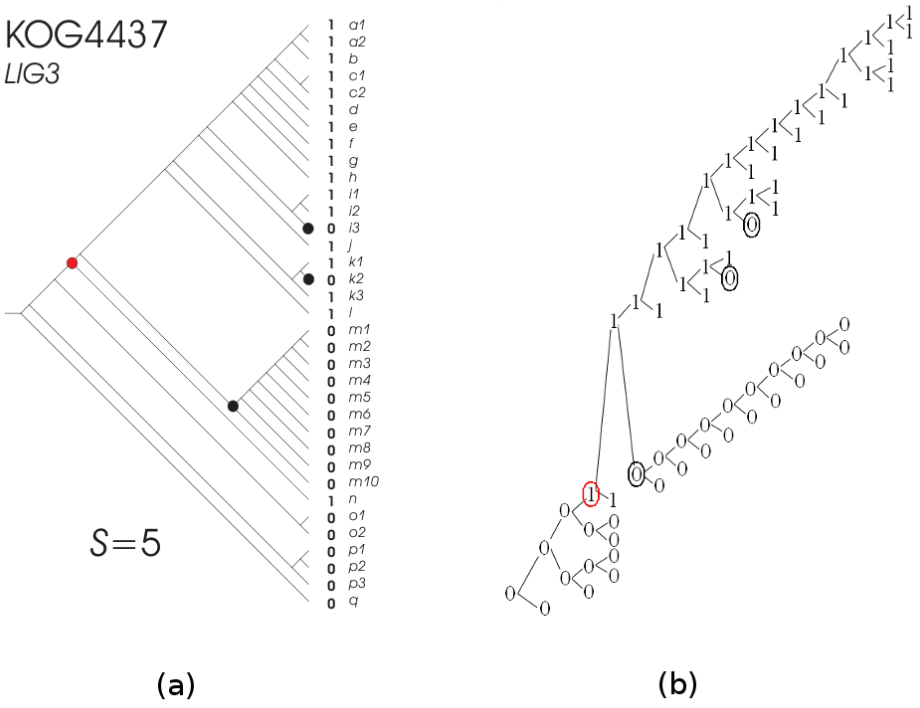

Figura 4.1: KOG4437 - LIG3

KOG0221. O valor de inconsistência encontrado, Figura (4.2b), $S = 9$, é o mesmo que foi encontrado via cálculo manual em [3], Figura (4.2a). Mas para esse KOG existe pelo menos uma degenerescência o que resultou em um padrão de presença/ausência diferente do obtido na Figura (4.2a), o qual claramente é o cenário mais parcimonioso pois embora possua um número maior de eventos, há somente um evento de surgimento do gene, enquanto que no padrão apresentado na Figura (4.2b) existem dois eventos de surgimento.

A modificação para definir o melhor cenário dentre vários com o mesmo valor mínimo de inconsistência será implementada na próxima versão do programa. Após esta correção o programa irá considerar como fator de

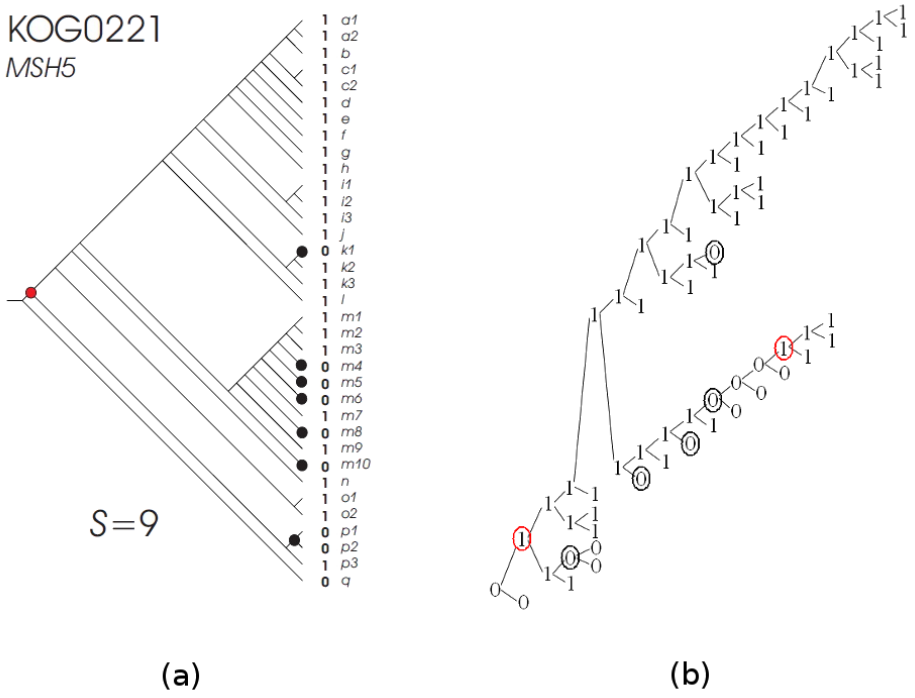

Figura 4.2: KOG0221 - MSH5

seleção para o cenário mais parcimonioso o número de eventos de surgimento de genes, e não somente do escore mínimo para o valor de inconsistência.

Capítulo 5

Considerações finais

A determinação da raiz evolucionária de grupos ortólogos individuais é importante porque possibilita a inferência da ancestralidade dos genes, propiciando a validação de medidas experimentais e fornecendo dados para a simulação do crescimento de redes de genes que possam ser utilizadas para direcionar o desenvolvimento dos modelos.

Os próximos passos envolvem o uso do programa para inferir a ancestralidade de todos os KOGs e considerando a árvore filogenética de todos os eucariotos disponíveis no STRING.

Algumas modificações ainda precisam ser implementadas como a contabilização dos eventos de surgimento e perda de genes, a automatização da modelagem da árvore e da leitura de KOGs. Tudo isso facilitará muito a utilização do programa e a determinação da ancestralidade para todos os KOGs, considerando que na versão atual os dados precisam ser inseridos

manualmente.

Além disso, novas formas de mapeamento do padrão de presença/ausência dos genes ortólogos nos nós internos da árvore podem ser pesquisadas, o que permitirá uma continuidade do estudo e consequentemente a otimização dos algoritmos.

Apêndice A

Programa

Nas próximas duas seções serão apresentados os códigos fonte do programa implementado. Na seção A.1, teremos o código *LCAFinder.java*, que é o aplicativo principal e na seção A.2 o código *Arvore.java* com as classes e métodos que modelam a árvore e executam todos os cálculos.

A.1 LCAFinder.java

```
public class LCAFinder {

  public static int nivelMaximoDaArvore;
  // número de configurações possíveis de valores dos nós internos
  public static int valoresPossiveis = 0;
  public static int nodesValuesArraySize;

  // array descrevendo árvore de consenso
```

```
  public static int treeArray[] =
  {2,1,8,4,12,3,6,10,14,5,7,9,11,13,34,16,36,15,18,35,42,17,20,38,
   44,19,22,37,40,43,50,21,24,39,41,46,52,23,26,45,48,51,54,25,28,
   47,49,53,56,27,30,55,58,29,32,57,60,31,33,59,64,62,66,61,63,65,
   68,67,69};

// Valores dos nós folha (o array corresponde a um padrão filético)
  public static int leafNodesArray[] =
// KOG4437 LIG3 - OK - Fechou com o artigo de 2008 - S = 5
  {0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0,1,1,0,1,1,1,1,1,1,1,1,1,
   1,1,1};


  public static void main( String args[] ) // método principal
  {

    Arvore tree = new Arvore(); // cria um objeto árvore vazio

    for ( int contador = 0; contador < treeArray.length; contador++)
    {
      tree.inserirNode( treeArray[ contador ] );
    }

//  Define o tamanho do vetor que armazenará os valores dos nós
//  internos da árvore = número de nós internos da árvore
    nodesValuesArraySize = treeArray.length - leafNodesArray.length;

// Determinando o nível máximo da árvore
    nivelMaximoDaArvore = tree.determinarNiveis();

// Chama método que determina os nós pai
    tree.setParents();

// Chama método que lê os valores dos nós folha
    tree.setLeafValues( leafNodesArray );

//====>  INÍCIO DAS SUB-ÁRVORES - cálculo dos nós internos

// Chama método que procura grupos de nós folhas com mesmos valores
// para definir os valores de seus nós pais de modo a diminuir o
// número de combinações de valores para os nós internos

//====// PRIMEIRA SUB-ÁRVORE
    nodesValuesArraySize= tree.simplifyNodesValues(36);
```

```
    System.out.println("");
    workWithSubTree(nodesValuesArraySize);
// Gera e varia os valores dos nós internos
    tree.setNodesValuesSimplified(valoresPossiveis);

//====// SEGUNDA SUB-ÁRVORE
    nodesValuesArraySize = tree.simplifyNodesValues(16);
    System.out.println("");
    workWithSubTree(nodesValuesArraySize);
    tree.setNodesValuesSimplified(valoresPossiveis);

//====// TERCEIRA SUB-ÁRVORE
    nodesValuesArraySize = tree.simplifyNodesValues(4);
    System.out.println("");
    workWithSubTree(nodesValuesArraySize);
    tree.setNodesValuesSimplified(valoresPossiveis);

//====> ÁRVORE COMPLETA - cálculo dos nós internos

//  Determina o número de nós internos após analisar as sub-árvores
    tree.findRemainingNodes();

// Chama método que exibe representação da árvore
    tree.showTree();

// Exibindo Resultados
    System.out.printf("\n############# RESULTADO #############\n");
    System.out.printf("\nMenor valor de inconsistência: S = %d\n",
Arvore.s);
    tree.getLCA(); // Método que exibe a posição do LCA
    System.out.printf("\n############ ############ ############\n\n");
  } // fim do método main

  public static void workWithSubTree(int nodesArraySize)
  {
// Define o número de configurações existentes
    for ( int i = 0; i < nodesArraySize; i++)
    {
      valoresPossiveis += (int) Math.pow(2, i );
    }
  } // fim do método workWithSubTree
} // fim da classe LCAFinder
```

A.2 Arvore.java

```
public class Arvore
{
  private NoDaArvore raiz;
  private static NoDaArvore nodeAux;
  public int nivelAux = 0;
  public int nivelMaximo = 0;
  public static int s = 5000;
  public static int sAux = 0;
  public static int identificadorConjuntoDados = 0;
  public final int G = 2; // peso de surgimento dos genes
  public final int L = 1; // peso da perda dos genes
  private static NoDaArvore lCANodeValue; //Último Ancestral Comum
  public static int contadorFolhas;
  public static int currentNodeIndex;
  public static int usableNodeCounter;
  public static int globalUsableNodeCounter = 0;
  public static int leafNodesArray[];
  public static int nodesValuesArray[];
  public static int nodesValuesArrayAux[];

  public Arvore()
  {
    raiz = null; // Cria nó inicial
  } // fim do construtor Arvore()

  public void inserirNode( int indiceInserido )
  {
    if ( raiz == null )
    {
      raiz = new NoDaArvore( indiceInserido );
    }
    else
    {
      raiz.inserir( indiceInserido );
    }
  } // fim do método inserirNode

  public int determinarNiveis()
```

```
{
  determinarNiveisPriv( raiz );
  return nivelMaximo;
} // fim do método determinarNiveis

private int determinarNiveisPriv( NoDaArvore node )
{
  nivelAux++;
  if ( node == null )
  {
    nivelAux--;
    return 0;
  }
  determinarNiveisPriv( node.nE ); // percorre subárvore esquerda
  determinarNiveisPriv( node.nD ); // percorre subárvore direita

  nivelAux--;
  node.nivel = nivelAux; // atribui nivelAux ao nível do nó

  // determina maior nível da árvore
  if ( nivelAux > nivelMaximo )
      nivelMaximo = nivelAux;

  // Retorna o valor do maior nível
  return nivelMaximo;
} // fim do método determinarNiveisPriv

public void setLeafValues( int valorFolhas[] )
{
  leafNodesArray = valorFolhas;
  contadorFolhas = 0;

  setLeafValuesPriv( raiz );
} // fim do método setLeafValues

private void setLeafValuesPriv( NoDaArvore node )
{
  if ( node == null )
    return;

  setLeafValuesPriv( node.nE ); // percorre subárvore esquerda
  setLeafValuesPriv( node.nD ); // percorre subárvore direita

  if ( node.nE == null & node.nD == null )
```

```
  {
      // variavel informando que este nó é um nó folha
      node.leaf = true;
      if ( contadorFolhas < leafNodesArray.length )
      {
        node.valor = leafNodesArray[ contadorFolhas ];
        contadorFolhas++;
      }
  }
  return;
} // fim do método setLeafValuesPriv

public int simplifyNodesValues(int indiceDoNo)
{
  usableNodeCounter=0;
  simplifyNodesValuesPriv(raiz, indiceDoNo);
  nodesValuesArray = new int[usableNodeCounter];
  nodesValuesArrayAux = new int[usableNodeCounter];
  return usableNodeCounter;
} // fim do método simplifyNodesValues

private void simplifyNodesValuesPriv(NoDaArvore node, int indiceDoNo)
{
  if (node == null)
    return;

  node.usableNode = false;

  if (node.indice == indiceDoNo)
  {
    node.usableNode = true;
    node.subRoot = true;
    usableNodeCounter++;
  }

  if (node != raiz /*& node.indice != indiceDoNo*/)
  {
    if (node.nP.usableNode)
    {
      node.usableNode = true;
      if (!node.leaf)
      {
        usableNodeCounter++;
      }
```

```
      }
    }
    simplifyNodesValuesPriv(node.nD, indiceDoNo);
    simplifyNodesValuesPriv(node.nE, indiceDoNo);
  } // fim do método simplifyNodesValuesPriv

  public void setNodesValuesSimplified(int nodesConfigValue)
  {
    int configuracoesPossiveis = (int) Math.pow(2, usableNodeCounter);

//  Cálculo de uma determinada configuração de valores para os nós internos
    for(int currentNodeConfigValue = 0;
currentNodeConfigValue < configuracoesPossiveis;
currentNodeConfigValue++)
    {
        nodesValuesArray = gerarValoresDosNodos(currentNodeConfigValue);
        currentNodeIndex = 0;

        setNodesValuesSimplifiedPriv(raiz);

// Calcula o valor de inconsistência s
        getInconsistencySimplified();

        if ( sAux < s)
        {
          s = sAux;
          identificadorConjuntoDados = currentNodeConfigValue;
        }
        currentNodeIndex = 0;
    }

// Gera array de valores internos para a configuração de menor s
  nodesValuesArray = gerarValoresDosNodos(identificadorConjuntoDados);

// Define os valores dos nós internos para a configuração do menor s
  setNodesValuesSimplifiedPrivFinal(raiz);

// Retorna o valor da inconsistência s para o valor inicial
  s = 5000;
  } // fim do método setNodesValuesSimplified

  private void setNodesValuesSimplifiedPriv(NoDaArvore node)
  {
    if (node == null)
```

```
      return;

    setNodesValuesSimplifiedPriv(node.nD);

    if (node.usableNode & !node.leaf)
    {
      node.valor = nodesValuesArray[currentNodeIndex];

      currentNodeIndex++;
    }

    setNodesValuesSimplifiedPriv(node.nE);
  } // fim do método setNodesValuesSimplifiedPriv

  private void setNodesValuesSimplifiedPrivFinal(NoDaArvore nodeAux)
  {
    if (nodeAux == null | nodeAux.leaf)
      return;

    setNodesValuesSimplifiedPrivFinal(nodeAux.nD);

    if (nodeAux.usableNode == true & nodeAux.locked == false)
    {
      nodeAux.valor = nodesValuesArray[currentNodeIndex];
      currentNodeIndex++;
      if (nodeAux.subRoot == false)
      {
        nodeAux.locked = true;
      }

// retorna o node para usable = false após fixar valores dos nós
      nodeAux.usableNode = false;
    }
    setNodesValuesSimplifiedPrivFinal(nodeAux.nE);
  } // fim do método setNodesValuesSimplifiedPrivFinal

  public void findRemainingNodes()
  {
    findRemainingNodesPriv(raiz);

    int numberOfCombinations = (int) Math.pow(2, globalUsableNodeCounter);

    nodesValuesArray = new int[globalUsableNodeCounter];
    nodesValuesArrayAux = new int[globalUsableNodeCounter];
```

```
    for (int currentNodeConfigValue = 0;
currentNodeConfigValue < numberOfCombinations;
           currentNodeConfigValue++)
    {
      currentNodeIndex = 0;
      nodesValuesArray = gerarValoresDosNodos(currentNodeConfigValue);

      setNodesValuesPriv(raiz);

      // Calcula valor de inconsistência
      getInconsistency();

      if ( sAux < s)
      {
        s = sAux;
        identificadorConjuntoDados = currentNodeConfigValue;
      }
      // retorna o contador de nós para 0
      currentNodeIndex = 0;
    }

    // => Gera array de valores internos para a configuração de menor s
    nodesValuesArray = gerarValoresDosNodos(identificadorConjuntoDados);

    // Define os valores dos nós internos para a configuração do menor s
    setNodesValuesPriv(raiz);
  } // fim do método findRemainingNodes

  private void findRemainingNodesPriv(NoDaArvore nodeAux)
  {
    if (nodeAux == null)
      return;

    findRemainingNodesPriv(nodeAux.nD);
    findRemainingNodesPriv(nodeAux.nE);

     if (nodeAux.leaf == false & nodeAux.locked == false)
     {
        globalUsableNodeCounter++;
     }
  } // fim de findRemainingNodesPriv

  private void setNodesValuesPriv(NoDaArvore nodeAux)
```

```
{
  if (nodeAux == null)
    return;

  setNodesValuesPriv(nodeAux.nD);

  if (nodeAux.leaf == false & nodeAux.locked == false )
  {
    nodeAux.valor = nodesValuesArray[currentNodeIndex];
    currentNodeIndex++;
  }
  setNodesValuesPriv(nodeAux.nE);
} // fim do método setNodesValuesPriv

public void setParents()
{
  nodeAux = raiz;
  setParentsPriv( raiz );
} // fim do método setParents

private void setParentsPriv( NoDaArvore node )
{

  if ( node == null )
    return;

  if ( node != raiz )
    node.nP = nodeAux;

  if ( node.nE != null )
  {
    nodeAux = node;
    setParentsPriv( node.nE );
  }

  nodeAux = node.nP;

  if ( node.nD != null )
  {
    nodeAux = node;
    setParentsPriv( node.nD );
  }
} // fim do método setParentsPriv
```

```
public void getInconsistencySimplified()
{
  sAux = 0;
  getInconsistencySimplifiedPriv( raiz );
} // fim do método getInconsistencySimplified

private void getInconsistencySimplifiedPriv( NoDaArvore node )
{
  if ( node.nE != null )
  {
    getInconsistencySimplifiedPriv( node.nE );
  }

  if (node.subRoot == false)
  {
    if ( node.nP != null & node.usableNode)
    {
      if ( node.valor != node.nP.valor )
      {
        sAux = sAux +
        G*node.valor*(node.nP.valor-node.valor)*(node.nP.valor-node.valor)+
        L*node.nP.valor*(node.nP.valor-node.valor)*(node.nP.valor-node.valor);
      }
    }
  }

  if ( node.nD != null )
  {
    getInconsistencySimplifiedPriv( node.nD );
  }
} // fim do método getInconsistencySimplifiedPriv

public void getInconsistency()
{
  sAux = 0;
  getInconsistencyPriv( raiz );
} // fim do método getInconsistency

private void getInconsistencyPriv( NoDaArvore node )
{
  if ( node.nE != null )
  {
    getInconsistencyPriv( node.nE );
  }
```

```
  if ( node.nP != null )
  {
    if ( node.valor != node.nP.valor )
    {

      sAux = sAux +
      G*node.valor*(node.nP.valor-node.valor)*(node.nP.valor-node.valor)+
      L*node.nP.valor*(node.nP.valor-node.valor)*(node.nP.valor-node.valor);
    }
  }
  else
  {
    if ( node.valor == 1 ){
        sAux+=2; // surgimento do gene na raiz
    }
  }

  if ( node.nD != null )
  {
    getInconsistencyPriv( node.nD );
  }
} // fim do método getInconsistencyPriv

public int[] gerarValoresDosNodos(int valorConfiguracaoAux )
{
  int valorConfiguracao = valorConfiguracaoAux;

  int j = 0;
  int max = nodesValuesArray.length;
  int x = 0;
  int numero = valorConfiguracao;

  do
  {
    x = numero / 2;
    nodesValuesArrayAux[j] = numero % 2;
    numero = x;
    j++;
  }while ( j < max );

  return nodesValuesArrayAux;
} // fim do método public void gerarValoresDosNodos
```

```
public void showTree( )
{
  System.out.println("");
  System.out.println("\n########## REPRESENTAÇÃO DA ÁRVORE ##########\n");
  showTreePriv( raiz , 0 );
} //fim do método showTree

private void showTreePriv( NoDaArvore node, int contSpaces )
{
  int totalSpaces = contSpaces;
  if ( node == null )
    return;
  showTreePriv( node.nD , totalSpaces+5 );

  for ( int forCounter=1; forCounter <= totalSpaces; forCounter++ )
  {
    System.out.print( " " );
  }
  // Exibe índices e valores dos nós
  System.out.println( node.indice + " " +  node.valor );

  showTreePriv( node.nE , totalSpaces+5 );
  totalSpaces+=3;
} // fim do método showTreePriv

 public void getLCA()
{
  lCANodeValue = new NoDaArvore(100);
  lCANodeValue.nivel = 500;
  getLCAPriv( raiz );
  System.out.printf("LCA corresponde ao nó de índice: %d\n",
                    lCANodeValue.indice);
} // fim do método getLCA

private void getLCAPriv(NoDaArvore nodeAux)
{
  if (nodeAux == null)
    return;

  if (nodeAux == raiz )
  {
    if (nodeAux.valor == 1)
      lCANodeValue = nodeAux;
  }
```

```
    else
    {
      if (nodeAux.leaf)
      {
        return;
      }

      if(nodeAux.valor == 1 & nodeAux.nP.valor == 0)
      {
        if (nodeAux.nivel < lCANodeValue.nivel)
        {
          lCANodeValue = nodeAux;
        }
      }
    }

    getLCAPriv(nodeAux.nD);
    getLCAPriv(nodeAux.nE);
  } // fim do método getLCAPriv

} // fim da classe Arvore

class NoDaArvore
{
  public NoDaArvore nD; // nó direito
  public NoDaArvore nE; // nó esquerdo
  public NoDaArvore nP; // nó pai
  public int indice;
  public int nivel;
  public int valor;
  public boolean leaf = false;
  public boolean locked = false;
  public boolean subRoot = false;
  public boolean usableNode = false;

  // contrutor que inicializa um novo nó da árvore
  public NoDaArvore( int nodeData )
  {
    indice = nodeData;
    nD = nE = null;
  } // fim do construtor NoDaArvore

  public void inserir( int indiceInserido )
  {
```

```
    if ( indiceInserido < indice )
    {
      if ( nE == null )
      {
        nE = new NoDaArvore( indiceInserido );
      }
      else
      {
        nE.inserir( indiceInserido );
      }
    } // fim do if para nodo esquerdo

    if ( indiceInserido > indice )
    {
      if ( nD == null )
      {
        nD = new NoDaArvore( indiceInserido );
      }
      else
      {
        nD.inserir( indiceInserido );
      }
    } // fim do elseif para nodo direito
  } // fim do método inserir
} // fim da classe NoDaArvore
```

Referências Bibliográficas

[1] MIRKIN, B. G., FENNER, T. I., GALPERIN, M. Y. and KOONIN, E. V., 2003, "Algorithms computing parsimonious evolutionary scenarios for genome evolution, the last universal common ancestor and dominance of horizontal gene transfer in the evolution of prokaryotes", BMC Evol. Biol.,3.

[2] KOONIN, E. V., 2005, "Orthologs, Paralogs, and Evolutionary Genomics", Annu. Rev. Genet., 39, 309-338.

[3] CASTRO, M. A. A., DALMOLIN, J. S. R., MOREIRA, J. C. F., MOMBACH, J. C. M., DE ALMEIDA, R. M. C., 2008, "Evolutionary origins of human apoptosis and genome-stability gene networks", Nucleic Acids Research, Vol 36, 6269-6283.

[4] JENSEN, L. J., KUHN, M., STARK, M., CHAFFRON, S., CREEVEY, C., MULLER, J., DOERKS, T., JULIEN, P., ROTH, A., SIMONOVIC, M., BORK, P., VON MERING, C., "STRING 8–a global view on proteins and their functional interactions in 630 organisms", Nucleic Acid Research, Vol 37, Special Issue, pp D412-D416, January 2009.

[5] DEITEL, H. M., DEITEL, P. J., "Java: como programar", 2005, Pearson Prentice Hall, 6^a edição.

[6] CICCARELLI, F.D., DOERKS, T., VON MERING, C., CREEVEY, C.J., SNEL, B. AND BORK, P., 2006, "Toward automatic reconstruction of a highly resolved tree of life", Science, 311, 1283–1287.

[7] LETUNIC, I. AND BORK, P., 2007, Interactive Tree Of Life (iTOL): an online tool for phylogenetic tree display and annotation. Bioinformatics, 23, 127–128.

[8] PENNISI, E., 2003, "Drafting a tree", Science, 300, 1694.

[9] BALDAUF, S.L., 2003, "The deep roots of eukaryotes", Science, 300, 1703–1706.

[10] KATINKA, M.D., DUPRAT, S., CORNILLOT, E., METENIER, G., THOMARAT, F.,PRENSIER, G., BARBE, V., PEYRETAILLADE, E., BROTTIER, P., WINCKER, P. et al., 2001, "Genome sequence and gene compaction of the eukaryote parasite Encephalitozoon cuniculi", Nature, 414, 450–453.

[11] DELSUC, F., BRINKMANN, H. AND PHILIPPE, H., 2005, "Phylogenomics and the reconstruction of the tree of life", Nat. Rev. Genet.,6, 361–375.

[12] KUNIN, V., OUZOUNIS, C.A., 2003, "The balance of driving forces during genome evolution in prokaryotes", Genome Res, 13:1589-1594.

[13] SNEL, B., BORK, P., and HUYNEN, M.A., 2002, "Genomes in Flux: The evolution of archaeal and proteobacterial gene content", Genome Res, 12:17-25.