UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE BIOQUÍMICA E IMUNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM BIOINFORMÁTICA

FERRAMENTAS E SERVIÇOS ONLINE PARA A ANÁLISE DA ORIGEM CLADÍSTICA DE GENES E VIAS METABÓLICAS

HENRIQUE VELLOSO FERREIRA MELO

BELO HORIZONTE
2014

HENRIQUE VELLOSO FERREIRA MELO

FERRAMENTAS E SERVIÇOS ONLINE PARA A ANÁLISE DA ORIGEM CLADÍSTICA DE GENES E VIAS METABÓLICAS

Documento submetido ao Programa de Pós-Graduação em Bioinformática da Universidade Federal de Minas Gerais, como parte dos requisitos para a defesa de tese do programa.

Orientador:

Dr. José Miguel Ortega

BELO HORIZONTE

2014

AGRADECIMENTOS

A meu pai Cláudio e meu irmão Marcelo, por terem me apoiado em todos os momentos com carinho incondicional.

A meus avós Laura, Eneida, José Melo - exemplos de caráter e dignidade - e a todos os demais membros de minha família pelo afeto e apoio.

À Tiça, pelo carinho maternal, e à Carol, minha eterna irmãzinha, pelo apoio e carinho nos momentos mais difíceis.

Aos colegas de trabalho e amigos da Tokenlab, em especial a meus sócios Tiago Gaspar e Ricardo Almeida: pela compreensão nos momentos difíceis, amizade, apoio e por compartilharem comigo os grandes desafios dessa jornada.

A meus amigos de São Carlos: Bruno Perin e Alex Born (pelas dicas de PNL), Victor Stábile, Pablo Bizzi, Thomás Capiotti e todos os demais pelas dicas, apoio e compreensão.

Aos colegas, professores e funcionários do Programa de Pós-Graduação em Bioinformática da UFMG, pela cooperação e companheirismo.

A Héctor Urbina, Thomaz Perez-Acle e todos os amigos do Chile pela grande contribuição que deram para a realização deste trabalho (plataforma BOWS).

Aos colegas do Laboratório de Biodados da UFMG: Henrique Assis, Tetsu Sakamoto, Lucas Ferreira, Katia Lopes, Elisa Donnard, Adriano Barbosa-Silva, Fernanda Stussi, Luiza Quadros. Todos contribuíram Em meio a tantas adversidades, a realização deste trabalho não seria possível sem o auxílio deles. Agradecimentos especiais a Gabriel Fernandes, por fornecer a base de dados UEKO, e a Ricardo Vialle, por contribuir com os clientes para a plataforma BOWS.

Agradecimentos especiais a Verônica Melo e Rafael Guedes por caminharem juntos e darem forças nos momentos mais difíceis, além de contribuir ativamente para a realização deste trabalho.

Agradecimento especial a meu orientador José Miguel Ortega, pela tutoria, incentivo, compreensão e por estar sempre presente quando preciso.

À CAPES, da qual fui bolsista no início do período de realização deste trabalho. Os recursos fornecidos foram imprescindíveis para a realização do mesmo.

A todos que estiveram envolvidos direta ou indiretamente na realização deste trabalho.

Agradecimento especial à minha mãe, que mesmo em um plano distante, é e sempre será meu maior exemplo de vida; pois se hoje sou alguém com dignidade, honestidade e caráter, foi ela quem plantou as sementes.

SUMÁRIO

1. INTRODUÇÃO 1

1.1 MOTIVAÇÃO 5
1.2 OBJETIVOS 5
1.3 ORGANIZAÇÃO 6

2. DEFINIÇÕES IMPORTANTES 7

2.1 A LINGUAGEM DE PROGRAMAÇÃO *GROOVY* 7
2.2 O GERENCIADOR BANCO DE DADOS *MYSQL* 8
2.3 *WEB SERVICES* 8
2.4 NCBI *TAXONOMY* 9
2.5 BANCO DE DADOS KEGG 10
2.6 BASE DE DADOS UEKO 10

3. MATERIAIS E MÉTODOS 13

3.1 CARREGAMENTO DA BASE TAXONÔMICA DO NCBI NO BANCO DE DADOS LOCAL 13
3.2 CARREGAMENTO DA ÁRVORE TAXONÔMICA DO NCBI EM MEMÓRIA RAM 14
3.3 CONSTRUÇÃO DA TABELA *TAXSIMPLE* 15
3.4 IMPLEMENTAÇÃO DO MÓDULO LCA 16
3.5 IMPLEMENTAÇÃO DOS *WEB SERVICES* 17
3.6 IMPLEMENTAÇÃO DO MÓDULO *UEKOLCA* 19
3.7 IMPLEMENTAÇÃO DO *SCRIPT* KGMLTODB 21
3.8 CONSTRUÇÃO DAS TABELAS *PATH_SUBGRAPH* E *SUBGRAPH_ELEMENT* 23
3.9 PASSO-A-PASSO DO ALGORITMO DE LCA MÚLTIPLO 25
3.9.1 *Lista positiva e lista negativa* *25*
3.9.2 *Construção da LCA Tree* *26*
3.9.3 *Detecção de múltiplos LCAs* *28*
3.9.4 *Construção da Mini LCA Tree* *29*
3.10 DESENVOLVIMENTO DO *WEBSITE* GENESIS 30
3.10.1 *KO Origin* *31*
3.10.2 *Species Origin* *32*
3.10.3 *Multi LCA Tree* *33*
3.11 BOWS – *BIOINFORMATICS OPEN WEB SERVICES* 34

4. FERRAMENTAS E SERVIÇOS TAXONÔMICOS 38

4.1 *TAXSIMPLE* 39
4.2 CARREGAMENTO DA ÁRVORE TAXONÔMICA EM MEMÓRIA 41
4.3 *GETALLCHILDREN* 42
4.4 LCA – *LOWEST COMMON ANCESTOR* 44
4.5 *GENEORIGIN* 47
4.6 PADRÕES DE SURGIMENTO DE GENES EM UMA ESPÉCIE 50
4.7 LCA DE FUNÇÕES MOLECULARES E PROCESSOS BIOLÓGICOS 51
4.8 LCA DOS GENES, FUNÇÕES E PROCESSOS EM OUTRAS ESPÉCIES 53
4.9 ORIGEM DAS VIAS METABÓLICAS HUMANAS 56
4.9.1 *Análises quantitativas do número de genes das vias* *58*
4.9.2 *Subvias* *59*
4.9.3 *Passo-a-passo do surgimento da via de sinalização JAK-STAT* *62*

4.10 Discussão 64

5. LCA MÚLTIPLO E GENESIS 66

5.1 LCA Múltiplo 66
5.1.1 *Visão geral do algoritmo de LCA Múltiplo* *69*
5.1.2 *Lista positiva e negativa* *70*
5.1.3 *LCA Tree* *70*
5.1.4 *Detecção dos LCAs* *74*
5.2 Genesis 78
5.2.1 *KO Origin (Origem dos genes)* *80*
5.2.2 *Species origin (origem das espécies)* *82*
5.2.3 *Árvore de múltipla origem* *86*
5.3 Estudos de Caso 90
5.4 Discussão 92

6. BOWS 94

6.1 Funcionamento de BOWS 95
6.2 Ciclo de BOWS 96
6.3 Transações de *Front-end* 98
6.4 Transações de *Back-end* e o agente *Arrow* 101
6.5 Autenticação e segurança 103
6.6 Registro de ferramentas no BOWS 104
6.7 Criação dos clientes de *Web service* 105
6.8 Instalação de BOWS 105
6.9 Estudo de caso 105
6.10 Discussão 108

7. CONCLUSÕES 110

8. BIBLIOGRAFIA 111

APÊNDICE A – TRECHOS DE CÓDIGO 116

Back-end da ferramenta Prank Align 116
Front-end da ferramenta Prank Align em Matlab 117
Front-end da ferramenta Prank Align em Java 118

APÊNDICE B – TRABALHO SUBMETIDO À REVISTA BIOINFORMATICS 118

LISTA DE FIGURAS

FIGURA 1 - ESTRUTURA DE UM TÁXON, COMO VISTA NO *WEBSITE* DO NCBI 10
FIGURA 2 - CONSTRUÇÃO DE UM GRUPO UEKO. ELIPSES COLORIDAS REPRESENTAM DOIS GRUPOS UNIREF50 DISTINTOS. ELMENTOS CUJO ALINHAMENTO NÃO ALCANCE 50% DE COBERTURA COM AQUELE PRESENTE NO KO NÃO SÃO RECRUTADOS. 12
FIGURA 3 - DIAGRAMA ENTIDADE-RELACIONAMENTO PARA A BASE TAXONÔMICA 13
FIGURA 4 - TABELA *TAX_SIMPLE* 16
FIGURA 5 - ESTRUTURA DA APLICAÇÃO *GRAILS* QUE DISPONIBILIZA OS *WEB SERVICES* LCA E *TAXONOMY* 19
FIGURA 6 - TABELAS *UEKO* E *HSA_UEKO_LCA* 21
FIGURA 7 - EXEMPLO DE UM ARQUIVO DESCRITOR DE VIAS METABÓLICAS KGML 22
FIGURA 8 - MODELOS FÍSICOS DAS TABELAS *PATH_ENTRY, ENTRY_RELATIONSHIP* E *ENTRY_ELEMENTS* 22
FIGURA 9 - MODELOS FÍSICOS DAS TABELAS *PATH_SUBGRAPH* E *SUBGRAPH_ELEMENT* 24
FIGURA 10 - DIAGRAMA ER DO BANCO DE DADOS DE BOWS 36
FIGURA 11 - EXEMPLO DE CONSULTA NA TABELA GERADA POR *TAXSIMPLE* 40
FIGURA 12 - EXEMPLO DE RETORNO DO SERVIÇO *GETALLCHIDREN* 44
FIGURA 13 - EXEMPLO DE LCA 45
FIGURA 14 - LOG DO PROGRAMA DE LINHA DE COMANDO *LCARUNNER* 45
FIGURA 15 - EXEMPLO DE ARQUIVO XML RETORNADO PELO WEB SERVICE REST 47
FIGURA 16 - LINHAS DA TABELA *HSA_UEKO_LCA* 49
FIGURA 17 - *VIEW HSA_ORIGIN_UEKO (DIVIDIDA EM 2 PARTES)* 50
FIGURA 18 – PADRÃO DE SURGIMENTO DE GENES EM *HOMO SAPIENS* 51
FIGURA 19 - *HOMO SAPIENS:* SURGIMENTO DE GRIPOS DE ORTÓLOGOS X FUNÇÕES MOLECULARES X PROCESSOS BIOLÓGICOS 53
FIGURA 20 – SURGIMENTO DE GENES EM *DROSOPHILA MELANOGASTER* 54
FIGURA 21 - SURGIMENTO DE NOVOS PROCESSOS E FUNÇÕES EM *D. MELANOGASTER* 55
FIGURA 22 - *ARABIDOPSIS THALIANA:* SURGIMENTO DE GENES 55
FIGURA 23 - *ORYZA SATIVA:* SURGIMENTO DE GENES 56
FIGURA 24 - *ESCHERICHIA COLI:* SURGIMENTO DOS GENES 57
FIGURA 25 - *HOMO SAPIENS:* HISTOGRAMA DO NÚMERO DE GRUPOS KO X NÚMERO DE VIAS 58
FIGURA 26 - *METAZOA:* HISTOGRAMA DO NÚMERO DE GRUPOS KO X NÚMERO DE VIAS 59
FIGURA 27 - PORCENTAGEM DE VIAS METABÓLICAS COMPLETAS AO LONGO DA EVOLUÇÃO HUMANA 61
FIGURA 28 - PORCENTAGEM DE COMPLETUDE AO LONGO DA EVOLUÇÃO PARA VIAS *SINGLE CORE* E *MULTIPLE CORE* 61
FIGURA 29 - VIA DE SINALIZAÇÃO JAK-STAT EM *EUKARYOTA* 62
FIGURA 30 - VIA DE SINALIZAÇÃO JAK-STAT EM *METAZOA* 63
FIGURA 31 - VIA DE SINALIZAÇÃO JAK-STAT EM *EUMETAZOA* 63
FIGURA 32 - VIA DE SINALIZAÇÃO JAK-STAT EM *COELOMATA* 64
FIGURA 33 - VIA DE SINALIZAÇÃO JAK-STAT EM *EUTELEOSTOMI* 64
FIGURA 34 - DESLOCAMENTO NO LCA ORIGINAL CAUSADO PELA OCORRÊNCIA DE TRANSFERÊNCIA LATERAL DE GENES EM UM GRUPO DE ORTÓLOGOS 67
FIGURA 35 - EXEMPLO DA *LCA TREE* 73
FIGURA 36 – PRIMEIRO CRITÉRIO PARA CARACTERIZAÇÃO DE LCA: NODO É POSITIVO E POSSUI PAI NEGATIVO 75
FIGURA 37 - SEGUNDO CRITÉRIO PARA CARACTERIZAÇÃO DE LCA: NODO É FILHO ÚNICO POSITIVO DE PAI MISTO 75
FIGURA 38 - TERCEIRO CRITÉRIO PARA CARACTERIZAÇÃO DE LCA: NODO MISTO COM MAIS DE UM FILHO VIÁVEL 76
FIGURA 39 - SETOR DA *MINI LCA TREE* CONSTRUÍDA PARA O GRUPO DE ORTÓLOGOS K00226 78
FIGURA 40 - TELA PRINCIPAL DO *WEBSITE* GENESIS 80
FIGURA 41 - CAMPO PARA ENTRADA DO IDENTIFICADOR DE UM GRUPO DE ORTÓLOGOS KO EM GENESIS 81
FIGURA 42 – EXEMPLO DE RESULTADO NO *WEBSITE* GENESIS EXIBINDO MÚLTIPAS ORIGENS DE UM GENE 81
FIGURA 43 - TELA DO *WEBSITE* GENESIS PARA CONSULTA DA ORIGEM DE TODOS OS GENES DE UMA ESPÉCIE 82
FIGURA 44 - RELATÓRIO DO *WEBSITE* GENESIS EXIBINDO AS ORIGENS DE TODOS OS GENES DE UMA ESPÉCIE 84

FIGURA 45 - RELATÓRIO DO *WEBSITE* GENESIS EXIBINDO GENES DE UMA ESPÉCIE POSSIVELMENTE ADQUIRIDOS POR TRANSFERÊNCIA HORIZONTAL 84
FIGURA 46 - GRÁFICO GERADO PELO *WEBSITE* GENESIS PARA O SURGIMENTO DE GENES EM *HOMO SAPIENS* 85
FIGURA 47 - RELATÓRIO DO *WEBSITE* GENESIS EXIBINDO A CONTAGEM DO SURGIMENTO DE GENES HUMANOS POR CLADO 86
FIGURA 48 - TELA DE ENTRADA DE DADOS PARA A *MULTILCA TREE* NO *WEBSITE* GENESIS 88
FIGURA 49 - RELATÓRIO DO *WEBSITE* GENESIS EXIBINDO OS LCAS ENCONTRADOS EM UMA ANÁLISE 89
FIGURA 50 - ÁRVORE OBTIDA PELO ALGORITMO DE LCA MÚLTIPLO EXECUTADO SOBRE O GRUPO DE ORTÓLOGOS DE JAK1 90
FIGURA 51 - RESULTADO DO LCA MÚLTIPLO PARA ANÁLISE DO GENE *MARR* 91
FIGURA 52 - ARQUITETURA DE FUNCIONAMENTO DE BOWS 96
FIGURA 53 - DIAGRAMA DE SEQUÊNCIA DA PLATAFORMA BOWS 97
FIGURA 54 - ENTRADA E SAÍDA DA FERRAMENTA PRANK NA PLATAFORMA BOWS 106

LISTA DE TABELAS

TABELA 1 - ETAPAS E PRODUTOS DO ALGORITMO DE LCA MÚLTIPLO 69
TABELA 2 - EXEMPLO DE LISTA DE LCAS ENCONTRADOS PARA O GENE DA ENZIMA PLD1_2 77
TABELA 3 - MÉTODOS DE *FRONT-END* DA PLATAFORMA BOWS 99
TABELA 4 - PARÂMETROS DO MÉTODO *SUBMITPROCESS* 99
TABELA 5 - CAMPO DO TIPO *SUBMISSIONPARAM* 99
TABELA 6 - CAMPOS DO OBJETO *PROCESSRESULTRESPONSE* 100
TABELA 7 - MÉTODOS DE *BACK-END* DA PLATAFORMA BOWS 101
TABELA 8 - PARÂMETROS DO MÉTODO *NEXTQUEUEDPROCESS* 102
TABELA 9 - PARÂMETROS DO MÉTODO *CHANGEPROCESSSTATUS* 102
TABELA 10 - PARÂMETROS DO MÉTODO *INSERTPROCESSRESULT* 103
TABELA 11 - PARÂMETROS DO MÉTODO *CREATEAPPLICATION* 104
TABELA 12 – CONFIGURAÇÃO DE UM PARÂMETRO DE PROCESSO 104

ABSTRACT

The origins of genes have always fascinated scientists, and the list of proposed mechanisms for the origin of new genes has progressively increased. The exploitation of such mechanisms would greatly benefit from a description of the scenario underlying the rise of genes throughout evolution. This work creates tools for the description of this scenario and assesses the emergence of genes on large scale, not in chronological time, but in a cladistic context. The generated tools, all available online, can be grouped into three products: (i) a set of tools and services named BioTools Service (ii) the Genesis website and (iii) a Web services platform called BOWS. BioTools Service is a suite of scripts and services targeted to process taxonomy data and infer the origin of genes. It consists of four byproducts: TaxSimple, GetAllChildren, LCA and GeneOriginUEKO. TaxSimple flattens the hierarchical taxonomic data from NCBI in order to facilitate taxonomic queries. GetAllChildren is a Web service that returns all leaf nodes descendants of a given clade in the tree of life. LCA (Lowest Common Ancestor) is a script that returns the common ancestor of two or more given nodes. GeneOriginUEKO can infer the origin of genes using UEKO clusters of orthologous (an enriched version of the KEGG ORTHOLOGY database). Graphs generated based on the results show wave patterns of human gene emergence with peaks in certain clades, such as Euteleostomi and Eutheria. It was also observed that some human metabolic pathways were increasingly gaining elements throughout evolution, as seen for the JAK-STAT signaling pathway. However, it was found that horizontal transfer events could cause the LCA to incorrectly move toward the root of the tree of life. To work around this issue, the multiple LCA was developed, which isolates horizontal transfers with the aid of complete genomes and thereby infer multiple origins of a gene. The Genesis website, built based on this algorithm, provides a graphical interface to the main results of the gene multiple origins analysis and also permits performing of analyses with custom input data. At least this work presents the BOWS platform, a system that centralizes and provides universal and secure access to bioinformatics tools installed on remote computational clusters. BOWS can be installed in any lab and is already being used successfully in Biodados Lab at UFMG.

RESUMO

A origem dos genes sempre fascinou os cientistas, e a lista de mecanismos propostos para a origem de novos genes tem aumentado. A exploração desses mecanismos se beneficiaria muito da descrição de um cenário de aparecimento dos genes ao longo da evolução. Este trabalho cria ferramentas para descrição desse cenário e estima o surgimento de genes em larga escala, não dentro do tempo cronológico, mas em um contexto cladístico. As ferramentas produzidas e disponibilizadas online podem ser agrupadas em três produtos: (i) o conjunto de ferramentas e serviços BioTools Service, (ii) o website Genesis e (iii) a plataforma de Web services BOWS. BioTools Service é um suíte de *scripts* direcionados para processar dados de taxonomia e para inferência da origem gênica e é composto por quatro subprodutos: TaxSimple, GetAllChildren, LCA e GeneOriginUEKO. TaxSimple planifica a os dados taxonômicos hierárquicos do NCBI com o intuito de facilitar as consultas taxonômicas. GetAllChildren é um serviço que retorna todos os nodos-folha da árvore da vida que são descendentes de um clado. LCA (*Lowest Common Ancestor*) é um *script* que retorna o ancestral comum de dois ou mais nodos. GeneOriginUEKO é capaz de inferir a origem de genes a partir de grupos de ortólogos da base UEKO (versão enriquecida da base KEGG ORTHOLOGY). Gráficos gerados com base nos resultados mostram um padrão ondulatório de emergência de genes humanos com picos em certos clados, como Euteleostomi e Eutheria. Verificou-se também que as vias metabólicas humanas vão ganhando elementos ao longo da evolução, como observado na via de sinalização JAK-STAT. Porém, foi constatado que eventos de transferência horizontal poderiam levar o LCA erroneamente em direção à raiz da árvore da vida. Para contornar essa questão, foi desenvolvido o algoritmo de LCA Múltiplo, o qual utiliza genomas completos para isolar as transferências horizontais e, assim, inferir múltiplas origens de um gene. O website Genesis, construído com base nesse algoritmo, provê interface gráfica para os principais resultados das análises de origem múltipla e também permite a realização de análises com dados customizados. Finalmente, foi criada a plataforma BOWS, um sistema que centraliza e provê acesso universal e seguro a ferramentas de bioinformática instaladas em *clusters* computacionais remotos. BOWS pode ser instalado em qualquer laboratório e já vem sendo usado com sucesso no Laboratório de Biodados da UFMG.

1. INTRODUÇÃO

A origem dos genes é um tema que sempre fascinou os cientistas. Apesar de datada em alguns bilhões de anos atrás, quando o planeta vivia o "Mundo do RNA" [1], a verdadeira origem dos genes primordiais ainda é um mistério. Existe a hipótese, inclusive, de que as primeiras bases nitrogenadas tenham surgido de meteoritos que caíram no planeta anteriormente à origem da vida [2].

Em contrapartida à dificuldade do estudo dos genes primordiais, um número crescente de estudos facilitados pela era genômica e baseados em sequencias genômicas existentes revelaram uma diversidade espantosa de mecanismos subjacentes ao surgimento de genes mais recentes [3].

Há algumas décadas atrás, a duplicação gênica era considerada o único mecanismo por trás da origem da maioria dos novos genes. Renomado geneticista, Susumu Ohno [4] enfatiza que a presença de uma segunda cópia de um gene abre oportunidades únicas na evolução, por permitir que uma das duas cópias adquira novas funcionalidades enquanto a outra cópia preserva a função ancestral. Ohno ainda diz que "todo novo gene deve ter surgido a partir de um gene já existente". Em seu trabalho, Jacob [5] afirma que "a probabilidade de que uma proteína funcional surja *de novo* pela associação randômica de aminoácidos é praticamente zero".

No entanto, somente recentemente os detalhes moleculares foram examinados, e vários outros mecanismos como a retroposição, a fusão gênica, a transferência horizontal e origem *de novo* a partir de regiões não-codificadoras foram propostos [3; 6]. Além das pequenas modificações genéticas de genes pré-existentes que levam a diferenças nas suas sequências e/ou atividades, novos genes com novas funções podem ter contribuído significativamente para a evolução dos traços fenotípicos da espécie [3; 7]. Zhou *et al.* [8] estimaram que aproximadamente 12% dos novos genes no subgrupo de *Drosophila melanogaster* podem ter surgido *de novo* a partir de DNA não-codificadora. Toll-Riera *et al.* [9] identificaram 15 genes como esses em primatas. Knowles e McLysaght [10] descobriram três genes de *Homo sapiens* que surgiram *de novo* em DNA não-codificadora desde a divergência entre humanos e chimpanzés. Portanto, como já era suposto há bastante tempo [4], agora não há mais dúvidas de que novos genes têm contribuído significativamente para a evolução dos organismos.

É de se esperar que o mesmo processo também se aplique à evolução de vias metabólicas, já que os genes são os elementos fundamentais das mesmas. De fato, o conceito

darwiniano de descendência com modificação mostrou ser aplicável também a vias metabólicas [11].

A origem de um gene pode ser determinada em termos cronológicos, como realizado nos trabalhos de Smith *et al* [12] e Peterson *et al* [13]. Por meio dessa abordagem, procura-se determinar de maneira mais precisa possível a data histórica em que os genes aparecerem pela primeira vez. Para isso, se faz uso, dentre outras técnicas, da análise do relógio molecular e datação de registros fósseis. A abordagem proposta neste trabalho, no entanto, investiga o surgimento de genes não dentro do tempo cronológico, mas em um contexto evolutivo e cladístico. De fato, a análise cladística pode ser bastante informativa ao se inferir a história do desenvolvimento metabólico dentro de um contexto comparativo. Donnard *et al*. [14], por exemplo, estudaram a ancestralidade da via regulatória do desenvolvimento embrionário pré-implantação e descobriram que uma fração ancestral da via, incluindo os genes Nanog e Sox2, originou-se antes de *Chordata*, enquanto uma fração moderna, incluindo os genes Oct4 e LIF, apareceu na origem dos organismos placentários (*Eutheria*).

As ferramentas bioinformáticas de alinhamento e anotação disponíveis atualmente, aliadas ao alto desempenho de clusters de computadores e ao grande volume de sequencias disponível nas bases de dados públicos, permitiram a criação de bancos de dados de genes ortólogos como o *KEGG Orthology* [15]. Um grupo de ortólogos KO consiste em um conjunto de proteínas de diferentes organismos que se originaram a partir de um mesmo gene ancestral. Encontrar a origem desse gene na árvore da vida se resume a inferir o ancestral comum dos organismos representados no seu grupo de ortólogos KO. O NCBI (*National Center for Biological Information*) [16] fornece em seu servidor de FTP uma versão completa e atualizada de sua árvore taxonômica. A origem dos genes na árvore da vida pode ser inferida a partir do cruzamento dos dados taxonômicos e a utilização de algoritmos para determinação do ancestral comum (LCA - *lowest common ancestor*) de grupos de ortólogos.

Neste trabalho foram realizadas inúmeras análises sobre a ancestralidade de genes e vias metabólicas em humanos e outras espécies utilizando-se a abordagem cladística. Os dados gerados foram organizados em bancos de dados e disponibilizados *online* por meio de três produtos, os quais são descritos a seguir.

O primeiro produto, chamado de *BioTools Service*, consiste de um conjunto de serviços *online* que disponibilizam aos pesquisadores consultas rápidas e simples a dados taxonômicos do NCBI e de ancestralidade gênica. Tais serviços tem o objetivo de servirem como ferramentas de fácil acesso aos pesquisadores, poupando esforços não diretamente

relacionados às suas pesquisas. *BioTools Service* é constituído de quatro subprodutos: *TaxSimple*, *GetAllChildren*, LCA e *GeneOriginUEKO*.

TaxSimple é uma tabela simplificada de banco de dados construída a partir da árvore taxonômica do NCBI. A estrutura hierárquica em que os arquivos de dados taxonômicos do NCBI são distribuídos dificultam até mesmo consultas simples do tipo "liste todas as espécies da classe *Mammalia*". *TaxSimple* "planifica" a estrutura hierárquica dos dados do NCBI em uma única tabela em que cada linha armazena a linhagem de um organismo, do seu nó até a raiz da árvore. Nas colunas, *TaxSimple* armazena apenas os principais *ranks* (família, filo, reino, etc.) da linhagem, descartando os demais. Os nós escolhidos foram os mais populosos, além dos clássicos.

Como *TaxSimple* é uma tabela simplificada, pode não atender o pesquisador que necessita da árvore completa. Para contemplar esses casos, foram criados serviços que mantém a árvore taxonômica completa em memória no servidor, permitindo rápido acesso à mesma. Um desses serviços, chamado *GetAllChildren*, retorna todos os nodos-folha (que geralmente representam uma espécie) da árvore que são descendentes de um determinado nodo passado por parâmetro (a classe *Mammalia*, por exemplo).

Outro serviço que utiliza a árvore completa em memória é o LCA (Lowest Common Ancestor), que retorna o ancestral comum de dois ou mais nodos passados por parâmetro. O serviço LCA está disponível para os pesquisadores como *script* ou por *Web service* SOAP e REST.

O algoritmo de LCA aplicado a nodos da árvore taxonômica do NCBI é um dos dois elementos fundamentais da pesquisa realizada neste trabalho. O segundo elemento é a base de genes ortólogos UEKO (*UniRef Enriched KEGG Orthology*), resultado do trabalho de doutorado de Gabriel Fernandes [17], que constitui uma versão enriquecida da base de dados KO (*KEGG Orthology*) [15]. A base de dados KO original possui principalmente genes de espécies que já tiveram seu genoma completamente sequenciado. A versão de Fernandes enriquece os grupos ortólogos originais com sequencias que fazem parte de *clusters UniRef50* [18], o que provê grupos mais representativos. O último subproduto de *BioTools Service*, denominado *GeneOriginUEKO*, consiste em um conjunto de *scripts* para a inferência da origem cladística dos grupos de genes ortólogos da base UEKO. Após a descoberta da origem dos genes individuais, a análise se estende para o entendimento do padrão de surgimento de novos genes em espécies. Os gráficos gerados mostram um padrão ondulatório na maioria dos organismos e evidenciam que em certos clados houve uma explosão de novos genes, como acontece em *Euteleostomi* e *Eutheria*.

Ainda com base nos dados de origem gênica, foram também construídos *scripts* para a análise do surgimento e desenvolvimento das vias metabólicas. Verificou-se que as vias vão ganhando elementos ao longo da evolução, como foi possível observar com a análise da via de sinalização *JAK-STAT*.

Os resultados obtidos com a utilização do algoritmo de LCA para a inferência da origem cladística de genes foram considerados satisfatórios para a maioria dos genes estudados. Porém, foi constatado um fenômeno biológico que poderia causar desvios significativos na descoberta da verdadeira origem de certos genes: as transferências horizontais (ou laterais). Este fenômeno, mais comum em procariotos, é um processo que envolve a transferência de material genético entre organismos não descendentes entre si. Desse modo, é possível que um gene originário em um ramo da árvore da vida esteja presente em organismos não irmãos desse ramo, fazendo com que o LCA suba em direção à raiz e consequentemente corrompendo o resultado das análises. Para contornar essa questão, foi desenvolvido neste trabalho o algoritmo de LCA Múltiplo, que utiliza o banco de genomas completos do UniProt [19] para encontrar as múltiplas ancestralidades de um gene na árvore da vida. Genomas completos que comprovadamente não possuem o gene estudado ajudam a dividir o LCA inicial em sub-LCAs. Um dos sub-LCAs encontrados representa a origem *de novo* do gene foco da análise, enquanto os demais indicam possíveis transferências horizontais ocorridas ao longo da evolução. O procedimento também é capaz de sugerir eventos de perda gênica.

A transferência lateral de genes entre organismos é tema de grande interesse da comunidade científica, pois é um forte mecanismo que molda o processo evolutivo. Com essa premissa foi desenvolvido e disponibilizado o *website* Genesis, o segundo produto deste trabalho. Genesis permite que pesquisadores explorem os principais resultados das análises de origem múltipla obtidos neste trabalho. O *site* disponibiliza três ferramentas principais: consulta à origem (ou múltiplas origens) de um grupo de ortólogos KO; análise do surgimento de genes em uma determinada espécie e as possíveis transferências horizontais ocorridas (incluindo o gráfico dos padrões de surgimento de genes); execução do algoritmo de LCA múltiplo em tempo real a partir da entrada de um conjunto de organismos que possuem um determinado gene (positivos) e um conjunto de organismos que não o possuem (negativos). Além de listar os múltiplos LCAs encontrados, o sistema exibe a *Mini LCA Tree*, uma representação visual dos resultados, contendo apenas os nodos relevantes que facilitam o entendimento.

Como último produto deste trabalho, foi desenvolvido BOWS, uma plataforma para centralização de *Web services* de Bioinformática. BOWS foi desenvolvido para auxiliar bioinformatas a disponibilizar, de uma forma universal e de fácil acesso, ferramentas e dados gerados em suas pesquisas. BOWS é uma plataforma de serviços remotos que utiliza a tecnologia de *Web services* para prover uma interface comum de entrada e saída a um conjunto heterogêneo de ferramentas de Bioinformática. Essas ferramentas podem ser executadas em *clusters* de alto desempenho, provendo à comunidade acesso a programas que de outra forma demandariam grande tempo de processamento. Além disso, BOWS provê acesso seguro aos programas que são executados em seu *back-end*, dificultando cópias e acessos não-autorizados. Modelado de forma genérica, BOWS suporta virtualmente qualquer ferramenta de bioinformática em seu *back-end*, e pela natureza universal de *Web services*, pode ser chamado a partir de qualquer linguagem de programação (que possua suporte a SOAP). Quando um cliente *front-end* é executado, submete o processo à plataforma BOWS e recebe um identificador. Todo o processo é executado pelo *back-end* cadastrado na plataforma BOWS de forma transparente. Quando o cliente requisita o resultado, este é enviado ao cliente se a execução já estiver concluída.

1.1 Motivação

Nos últimos anos, os pesquisadores vêm tentando entender melhor a maneira como surgem novos genes e isso está relacionado ao aparecimento de novos traços fenotípicos. Muito progresso tem sido alcançado principalmente na descoberta dos mecanismos moleculares que permitem o surgimento de novos genes. No entanto, a análise quantitativa do padrão e das frequências do surgimento de genes e vias metabólicas ao longo da evolução se faz necessária. É papel da bioinformática prover o dados e o ferramental necessário para que pesquisadores possam realizar suas análises subsequentes e entender melhor o processo evolutivo.

1.2 Objetivos

Este trabalho tem como objetivos principais:

- Gerar novas bases de dados com informações sobre o surgimento de genes e vias metabólicas.
- Gerar novas bases de dados sobre os padrões de surgimento de genes em espécies.

- Criar serviços e ferramentas que possibilitem consultas mais fáceis e rápidas aos dados taxonômicos do NCBI.
- Disponibilizar *Web services* para acesso programático aos dados e ferramentas gerados.
- Disponibilizar um *website* para acesso visual aos dados gerados.
- Auxiliar bioinformatas a disponibilizarem ferramentas e dados gerados em suas pesquisas de uma forma universal e prática.

1.3 Organização

Este documento está organizado da forma descrita a seguir.

No capítulo 2 são apresentadas definições importantes para o entendimento do trabalho.

No capítulo 3 são descritos os materiais, programas computacionais, implementações e técnicas desenvolvidas para a realização deste trabalho.

Os capítulos 4, 5 e 6 apresentam os produtos desse trabalho. O capítulo 4 apresenta ferramentas e serviços taxonômicos produzidos. O capítulo 5 descreve o algoritmo de LCA Múltiplo e o *website* Genesis. O capítulo 6, por sua vez, apresenta a plataforma BOWS. Ao fim de cada um desses capítulos há uma seção de discussão dos resultados.

O capítulo 7 apresenta as conclusões deste trabalho. Por fim, no capítulo 1 estão listadas as referências bibliográficas.

2. DEFINIÇÕES IMPORTANTES

Neste capítulo, são descritos alguns conceitos e tecnologias fundamentais para o entendimento deste trabalho.

A seção 2.1 apresenta a linguagem de programação *Groovy*, utilizada na implementação de todos os módulos e *scripts* deste trabalho.

O gerenciador de banco de dados *MySQL*, utilizado para armazenamento de dados e consultas, é apresentado na seção 2.2.

A seção 2.3 define o conceito de *Web services*, tecnologia utilizada com frequência neste trabalho para a disponibilização de ferramentas bioinformáticas para a comunidade.

O banco de dados *NCBI Taxonomy,* fundamental para as análises de ancestralidade realizadas neste trabalho, é apresentado na seção 2.4.

Outro elemento fundamental para a realização deste trabalho, o banco de dados de genes ortólogos KEGG, é descrito na seção 2.5.

Por último, a base UEKO, versão modificada da base de ortólogos KEGG ORTHOLOGY que é incrementada com proteínas do banco de dados UniProt, é descrita na seção 2.6.

2.1 A linguagem de programação *Groovy*

Neste trabalho, todos os *scripts* e módulos são construídos na linguagem de programação *Groovy* [20] (versão 2.1.7). Essa escolha se deve ao fato de *Groovy* ser uma linguagem moderna e dinâmica, incluindo diferenciais presentes em linguagens de *script (Perl, Pytho*n) tais como tipos dinâmicos, *closures* e DSL (*Domain specific languages)*, e ao mesmo tempo contando com o poder e a riqueza da biblioteca Java.

Groovy executa na Máquina Virtual Java e por isso também herdou sua natureza multiplataforma. Isso significa que a compilação do código-fonte *Groovy* gera um arquivo no formato *bytecode* Java que pode ser executado em diferentes sistemas operacionais sem nenhuma modificação. *Groovy* pode ser vista como uma versão aprimorada da linguagem Java, a qual reduz consideravelmente as linhas de código e garante um aumento efetivo da produtividade.

Além disso, para aplicações *Web, Groovy* conta a plataforma *Grails* [21] (versão 2.3.2). Baseado em "código por convenção", *Grails* praticamente elimina a necessidade de arquivos XML de configuração e integra de forma transparente poderosos *frameworks* como

Hibernate, *Spring* e *JQuery,* aumentando dramaticamente a produtividade em aplicações *Web.* Os *Web services* construídos neste trabalho foram todos implementados sobre a plataforma *Grails.*

2.2 O gerenciador banco de dados *MySQL*

O *MySQL Community Edition* [22] (versão 5.1.59) foi o sistema gerenciador de banco de dados escolhido para armazenamento dos dados e consultas neste trabalho. Essa versão de SGBD (Sistema Gerenciador de Banco de Dados) é gratuita, de código-livre e possui suporte a inúmeras funcionalidades. Entre elas, suporta múltiplos *engines,* como *MyISAM* e *InnoDB, stored procedures, triggers* e *views.*

MySQL permite a construção de bancos de dados relacionais e o uso completo da linguagem SQL para realização de consultas nos mesmos.

2.3 *Web services*

Web services oferecem um *framework* extensível para comunicação aplicação-aplicação, construídos a partir de protocolos *Web* existentes e baseados em formatos abertos e conhecidos.

Web services simplificam o processo de interação entre aplicações construídas em plataformas heterogêneas, definindo um mecanismo padrão para descrição, localização e comunicação entre aplicações *Web.* Por meio de *Web services,* uma aplicação expõe suas funcionalidades traduzidas em uma linguagem universal: o XML.

O *framework* de *Web services* pode ser dividido em três áreas: protocolos de comunicação, descrição de serviços e descoberta de serviços. Para cada uma das áreas são desenvolvidas especificações, as quais as mais utilizadas são:

- O protocolo SOAP (*Simple Object Access Protocol)* [23], que provê um padrão para comunicação entre *Web services*.
- A linguagem WSDL (*Web Services Description Language*) [24], que provê uma descrição formal para *Web services,* em um padrão universal (XML)*;*
- O UDDI (*Universal Description, Discovery, and Integration*) [24], que provê um diretório para consulta a descrições de *Web services* disponíveis.

Dessa forma, *Web services* são uma excelente tecnologia para a disponibilização de aplicações bioinformáticas para a comunidade. Neste trabalho, foram criados descritores

WSDL que permitem o uso de algumas das ferramentas criadas por meio do protocolo SOAP e também do protocolo REST [25].

2.4 NCBI *Taxonomy*

A base de dados taxonômicos do NCBI [16] é construída e mantida em uma tentativa de incorporar o conhecimento taxonômico e filogenético de uma variedade de fontes, tais como publicações da literatura, bancos de dados públicos da *Web* e recomendações de especialistas da área de taxonomia [26].

Os dados estão estruturados na forma de uma árvore de nodos, sendo cada nodo associado a uma unidade taxonômica, também chamada de táxon. Cada táxon é identificado por um identificador "*taxonomy id"* (ou "*tax id"*) numérico único. Além disso, possui um nome científico, um ou mais nomes populares, um *rank* e outras informações. O *rank* representa a classificação do clado a que o táxon faz parte (reino, família, classe, ordem, espécie, etc.). A Figura 1 demonstra a estrutura de um táxon, como vista no *website* do NCBI.

As folhas da árvore representam espécies existentes na atualidade ou extintas. Os nodos não-folha representam ancestrais comuns a todos os *taxa* representados nos nodos que originam a partir deles na árvore. O nodo raiz da árvore, portanto, representa o ancestral comum a todos os organismos existentes ou extintos.

O NCBI disponibiliza toda sua base taxonômica em um servidor FTP constantemente atualizado. A base foi carregada em um banco de dados local utilizando-se a metodologia explicada na seção 3.1 adiante.

Uma vez carregada em bancos de dados locais, é possível a realização de consultas simples a informações taxonômicas em linguagem SQL. Por exemplo, é possível descobrir todos os ancestrais diretos de um dado táxon com uma única linha de comando SQL, assim como os nomes científicos e populares de um conjunto de *taxa*.

Entretanto, os dados estão estruturados de forma recursiva, de forma que um nodo só possui informação sobre o seu nodo pai na árvore, sendo necessária uma operação recursiva para a navegação das folhas à raiz. Portanto, para consultas e associações de dados mais complexas, como as que envolvem buscas em profundidade na hierarquia da árvore, seriam necessárias uma série de requisições.

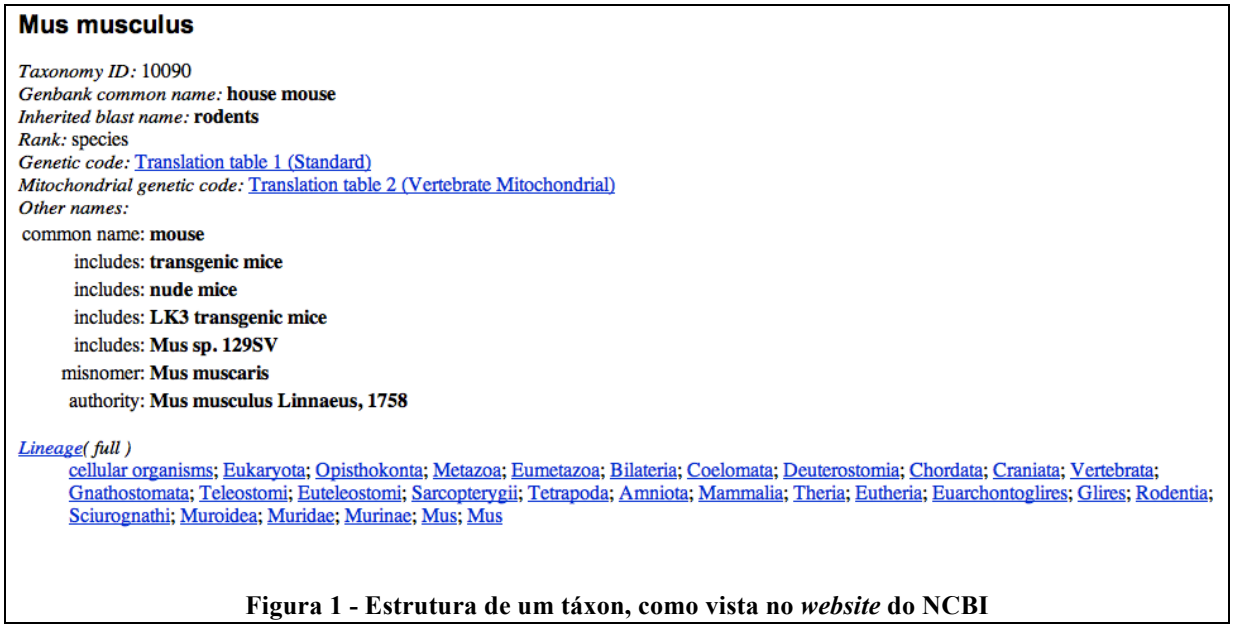


Figura 1 - Estrutura de um táxon, como vista no *website* do NCBI

2.5 Banco de dados KEGG

KEGG [15] é uma coleção de banco de dados constituída pela integração de informações genômicas, químicas e de sistemas. KEGG relaciona informações a nível molecular a aspectos funcionais de sistemas biológicos. Por isso, as informações contidas nessa base de dados são referência para interpretação biológica dos resultados de grandes projetos, principalmente os de sequenciamento.

KEGG possui uma série de bancos de dados categorizados. Dois desses bancos são de particular interesse neste trabalho. O banco de dados PATHWAY possui mapas de vias metabólicas e outros processos celulares. As vias metabólicas de PATHWAY são representadas de uma maneira geral, de modo a representar em uma única via todos os organismos que a possuem.

O banco de dados KO (KEGG ORTHOLOGY) consiste de grupos de ortólogos construídos manualmente. Cada grupo representa as variantes de um mesmo gene em diferentes organismos, isto é, variantes que tiveram origem em um ancestral comum. Os grupos são identificados por um número K, no formato K##### (no qual # representa dígitos). Uma vez associados a genes, os grupos KO são utilizados como nodos na construção de vias metabólicas PATHWAY.

2.6 Base de dados UEKO

O processo de anotação da base KEGG ORTHOLOGY utiliza unicamente genes de organismos com genoma totalmente sequenciado. Como o número de organismos com

genoma completo sequenciado ainda é relativamente pequeno, a identificação do ancestral comum que deu origem a um determinado gene de um grupo KO pode perder precisão. Por exemplo, se um grupo de ortólogos existe em uma espécie pertencente a um grupo que não tem nenhum genoma completo sequenciado, o ancestral comum correto pode estar mais próximo da raiz do que o calculado. Para aumentar a precisão na descoberta do ancestral comum, foi utilizado também o banco de dados UEKO (*UniRef Enriched KEGG Orthology*) [17], resultado do projeto de doutorado desenvolvido por Gabriel Fernandes no Laboratório de Biodados da UFMG. UEKO é construído por meio do enriquecimento dos grupos de ortólogos presentes nas bases KO a partir de sequências com alto grau de identidade (50%) E cobertura (50%) presentes na base de dados UniRef50 [18].

A base de dados UniRef50 agrupa sequencias com um percentual de identidade entre si superior a 50%. O algoritmo de montagem da base UEKO faz o recrutamento dessas sequencias para um dado grupo KO. Se um gene membro de um grupo KO possui sequencias presentes em um agrupamento UniRef50, todos os genes (proteínas Uniprot) membros do agrupamento UniRef50 que satisfaçam o limiar de 50% de cobertura do alinhamento são recrutados para esse seu grupo KO. O limiar de 50% de identidade já está contemplado pelo fato de todas integrarem o mesmo grupo UniRef50.

A Figura 2 mostra um esquema que exemplifica a maneira como os grupos UEKO são construídos. As estrelas representam genes. Nota-se que o grupo UEKO criado agrega todos os genes do grupo KO mais os genes de agrupamentos UniRef50 que satisfaçam o limiar de recrutamento.

O enriquecimento efetuado no momento da realização de muitos experimentos aqui descritos causa a elevação no número de ortólogos de 1.411.402 na base original KO para 2.881.880 em UEKO, um aumento de aproximadamente 104%. Além disso, pode-se então contar com ortólogos de organismos que até o momento não possuem o genoma completo sequenciado, o que aumenta a precisão na identificação da origem dos genes.

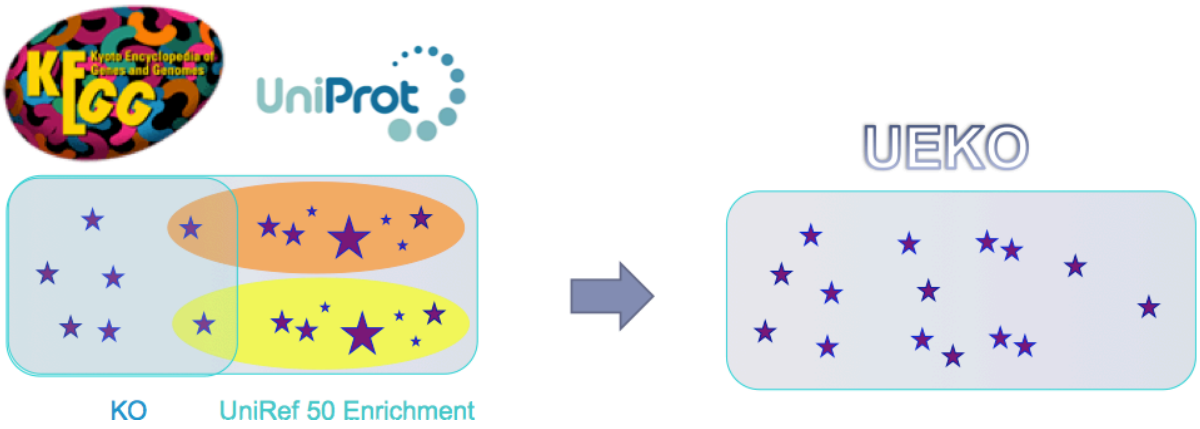


Figura 2 - Construção de um grupo UEKO. Elipses coloridas representam dois grupos UniRef50 distintos. Elmentos cujo alinhamento não alcance 50% de cobertura com aquele presente no KO não são recrutados.

3. MATERIAIS E MÉTODOS

Este capítulo descreve os materiais e métodos utilizados na produção deste trabalho. As seções apresentam detalhes técnicos de implementações de módulos e *scripts* construídos ao longo do projeto, com intuito de serem referenciadas a partir de seções dos capítulos 4, 5 e 6, onde se encontram os resultados deste trabalho.

3.1 Carregamento da base taxonômica do NCBI no banco de dados local

A base taxonômica completa do NCBI está disponível um arquivo compactado em seu servidor FTP, acessível pelo endereço ftp://ftp.ncbi.nih.gov/pub/taxonomy/taxdump.tar.gz. Dentro do arquivo compactado, existem dois arquivos principais: "*nodes*" e "*names*". *Nodes* contém informações sobre a hierarquia de nodos, enquanto *names* associa cada *tax id* a seus possíveis nomes (cientifico e populares).

Com base nos dados das colunas de interesse presentes nos arquivos *nodes* e *names,* foi criado um banco de dados local contendo duas tabelas: *tax_nodes* e *tax_names.* A Figura 3 apresenta o diagrama ER (entidade-relacionamento) desse banco de dados. Nota-se que as tabelas se relacionam através da coluna *tax_id,* que atua como chave primária em *tax_nodes* e chave estrangeira em *tax_names.* O relacionamento entre elas é de 1 para N, isto é, um nodo pode ter 1 ou mais nomes associados.

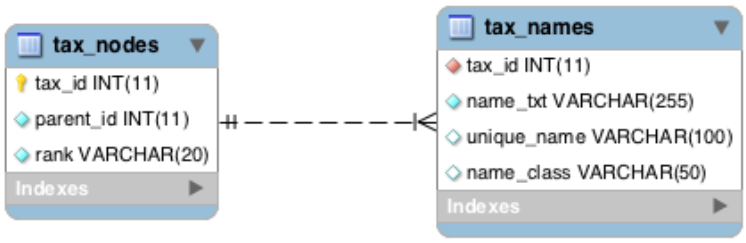


Figura 3 - Diagrama entidade-relacionamento para a base taxonômica

Os dados dos arquivos *names* e *nodes* estão disponíveis no formato TSV (valores separados por vírgula), o que permitiu o rápido carregamento dos mesmos no banco de dados local por meio do seguinte comando SQL:

```
LOAD DATA LOCAL INFILE <nome_arquivo> INTO TABLE <nome_tabela>
```

no qual <nome_arquivo> é o nome do arquivo a ser carregado (*nodes* ou *names)* e <nome_tabela> é o nome da tabela onde o arquivo será carregado (*tax_nodes* ou *tax_names).*

3.2 Carregamento da árvore taxonômica do NCBI em memória RAM

Para o carregamento da árvore taxonômica em memória, foi construído um módulo denominado *TaxonomyTree* na linguagem *Groovy*. O módulo também utiliza o *framework* Spring [27] para injeção de dependências.

O módulo *TaxonomyTree* possui três classes: *TaxonomyDAO, TaxonEntry*, e *TaxonomyTree*. *TaxonomyDAO* implementa o padrão de projetos DAO (*Data Access Object)*[28], sendo responsável por todas as transações com o banco de dados.

TaxonEntry representa os dados de um táxon da árvore taxonômica, possuindo os seguintes atributos:

- *taxId*: *id* do táxon.
- *parentId*: *id* do nodo pai do táxon.
- *name:* nome científico do táxon.
- *rank:* categoria do táxon (família, classe, gênero, etc.).
- *level:* distância em nodos do táxon até a raiz da árvore.

A classe *TaxonomyTree* armazena a árvore taxonômica em uma estrutura hierárquica, e possui métodos para a realização de operações sobre a mesma.

Assim que o módulo é iniciado, o método *mountTaxonomyTree* da classe *TaxonomyTree* é chamado. Esse método lê os dados das tabelas *tax_nodes* e *tax_names* do banco de dados local e a partir deles monta uma estrutura de árvore hierárquica em memória RAM.

A linguagem Java possui em sua API uma biblioteca bastante flexível para o armazenamento de estruturas em forma de árvore, a qual foi utilizada nesta implementação. Essa biblioteca se encontra no pacote *javax.swing.tree* da API Java (versão 1.4.2) e possui *DefaultMutableTreeNode [29]* como principal classe. Essa classe representa um nodo de uma árvore e possui ponteiros para o nodo pai e os nodos filhos. Pela utilização do conjunto de operações disponíveis, e pela natureza recursiva das árvores, é possível navegar por toda a árvore a partir de um único nodo. Cada nodo possui também referência a um *userObject,* que objetiva associar objetos do usuário aos nodos da árvore. Desse modo, nesta implementação cada nodo é associado a um táxon, através de um objeto da classe *TaxonEntry.*

O pseudo-código listado abaixo descreve a metodologia utilizada na montagem da árvore em memória.

```
Armazena todos os registros de tax_nodes em um vetor desordenado de objetos da classe TaxonEntry.
Cria o nodo raiz associado ao TaxonEntry nomeado root.
Para cada taxa presente no vetor desordenado:
    Se o nodo pai está presente na árvore:
        Cria um nodo para o filho associado ao seu respectivo TaxonEntry.
        Insere o nodo recém criado na árvore.
    Senão,
        Para cada táxon ancestral:
            Se o nodo pai está na árvore
                Insere toda a linhagem na árvore
```

Ao final do processo, todos os *taxa* estarão associados a um nodo na árvore e é possível navegar por toda a estrutura a partir de qualquer nodo.

3.3 Construção da tabela *TaxSimple*

A criação e o preenchimento da tabela *TaxSimple* são realizados por um módulo *Groovy* denominado *TaxSimple.* Esse módulo tem como dependência o módulo *TaxonomyTree* descrito na seção 3.2 acima, pois necessita do prévio carregamento da árvore em memória. O resultado de sua execução é a tabela *tax_simple,* a qual possui um registro para cada *taxa* contendo os *tax ids* dos principais *ranks* de sua linhagem até a raiz da árvore.

O módulo recebe como parâmetros, além dos bancos de dados de origem de dados e destino, uma lista de *unwanted ranks*, isto é, *ranks* que não estarão na tabela final. É importante notar que a tabela *tax_simple* é uma tabela simplificada, com um número fixo de colunas, por isso possui apenas os *ranks* mais significativos para cada organismo.

Neste trabalho, a lista de *unwanted ranks* inclui: *subkingdom*, *superphylum*, *infraclass*, *infraorder*, *parvorder*, *species group*, *species subgroup*, *tribe*, *subtribe*, *no rank*, *varietas* e *forma*. Esses *ranks* foram excluídos por não possuírem um número significativo de organismos associados a eles, ou por não representarem uma real classificação taxonômica (caso exclusivo de "no rank"). Por outro lado, os *ranks* que possuem colunas na tabela *tax_simple* são *superkingdom, kingdom, phylum, subphylum, superclass, class, subclass, superorder, order, suborder, superfamily, family, subfamily, genus, subgenus, species* e *subspecies.* A última coluna na tabela ainda guarda o *tax id* do genoma sequenciado, o qual

pode ser redundante com aquele da espécie, da subsespécie ou até mesmo distinto, se for associado a uma tribo, linhagem ou cêpa.

A Figura 4 apresenta o modelo da tabela *tax_simple*, incluindo todas as suas colunas.

A execução do módulo é simples. Faz-se uso de uma classe auxiliar denominada *SimpleTaxonEntry,* a qual possui um atributo para cada *rank* significativo. Esses atributos devem armazenar os *tax ids* dos *taxa* associados a esses *ranks.*

Primeiramente a árvore é carregada em memória pelo mesmo método descrito na seção 3.2. Em seguida, ela é totalmente percorrida em pós-ordem. Para cada nodo, um objeto da classe *SimpleTaxonEntry* é criado. Esse objeto é preenchido com os *tax ids* de toda a linhagem do táxon associado ao nodo, até a raiz da árvore. Um registro é inserido na tabela *tax_simple,* de modo que a coluna *genome_tax_id* é preenchida com o *tax id* do táxon e as demais colunas são preenchidas com os atributos de *SimpleTaxonEntry,* conforme os seus *ranks.* Ao fim do processamento, toda a árvore está acessível em *tax_simple.*

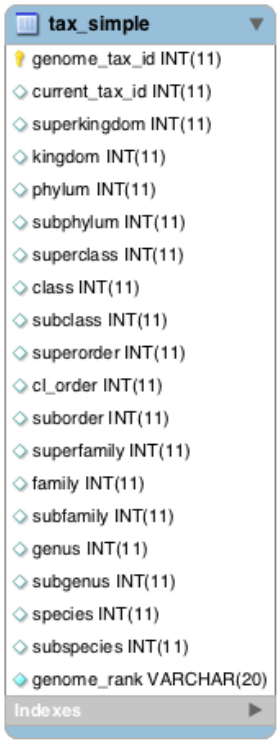


Figura 4 - Tabela *tax_simple*

3.4 Implementação do módulo LCA

Para a descoberta do ancestral comum de dois ou mais nodos da árvore taxonômica, foi construído um módulo na linguagem *Groovy* denominado LCA. Esse módulo é composto

por uma classe utilitária também denominada LCA e dois *scripts*: *LCARunner* e *MultiLCA.* A classe possui um método que de fato realiza o cálculo de LCA, e pode ser reutilizada como componente em outros módulos, como acontece no caso de *UekoLCA* e *GoaLCA*. Os *scripts* são utilizados para cálculo do LCA a partir de arquivos de entrada, via linha de comando.

O módulo LCA depende do módulo *Taxonomy* descrito na seção 3.2 para o carregamento da árvore em memória e as operações sobre ela. Assim que o módulo é executado, a árvore taxonômica é carregada em memória e fica acessível através de um objeto da classe *TaxonomyTree.*

A classe *LCA* possui apenas o método *lowestCommonAncestor,* o qual recebe como parâmetro uma lista de *tax ids* de tipo inteiro. A lista é passada como parâmetro no método *findNodeMapByTaxIds* do objeto *Taxonomy Tree*, o qual retorna um mapa de nodos em que a chave de cada registro é um *tax id* e o valor é o nodo na árvore taxonômica associado.

A classe *DefaultMutableTreeNode* (que é a classe dos nodos da árvore taxonômica, como descrito na seção 3.2) possui um método denominado *getSharedAncestor,* que retorna o ancestral comum entre dois nodos. Não há um método que calcule o ancestral comum para mais de dois nodos ao mesmo tempo, por isso, algoritmo descrito a seguir é utilizado.

Um nodo do mapa é escolhido de forma randômica para ser o LCA temporário inicial. Então, cada um dos outros nodos do mapa são comparados com o LCA temporário, a fim de saber se é descendente dele ou não. Se não for descendente, chama-se o método *getSharedAncestor* a fim de se determinar o ancestral comum entre os dois, o qual será o novo LCA temporário. O último LCA temporário encontrado é o LCA definitivo do grupo de nodos associados aos *tax ids* de entrada.

3.5 Implementação dos *Web services*

A implementação e disponibilização de *Web services* neste trabalho foi realizada sobre a plataforma *Grails. Grails* simplifica bastante essa tarefa pois constrói automaticamente uma aplicação *Web* pronta, tornando necessário apenas adicionar as bibliotecas e implementar os métodos de negócio. Além disso, é necessário o uso de um *plugin* para construção de *Web services*. No caso deste projeto, o *plugin* utilizado foi a extensão para *Grails* da biblioteca Apache CXF [30].

Neste trabalho foi criada uma aplicação *Web* denominada *BioToolsService,* a qual disponibiliza dois *Web services: LCA e Taxonomy.* Ambos os serviços dependem do módulo *TaxonomyTree* e o *Web service* LCA depende do módulo LCA. Esses módulos são inseridos

na aplicação por injeção de dependência, funcionalidade do *framework Spring* integrado. Assim que a aplicação é iniciada, a árvore taxonômica é carregada e disponibilizada em um *Spring bean,* podendo então ser acessada a partir qualquer classe.

O *Web service* LCA está disponível tanto através do paradigma REST quanto do protocolo SOAP. Para a disponibilização do mesmo por meio do protocolo SOAP foi criada uma classe de serviço denominada *LCAService* dentro do diretório *services* da estrutura da aplicação (ver Figura 5). Esta classe é marcada com a diretiva `expose=['cxf'].` Essa diretiva é lida pelo *plugin* CXF, o qual gera um arquivo WSDL para a classe de modo a expor todos os métodos públicos da mesma como um *Web service* SOAP.

Um método denominado *lca* foi criado dentro da classe *LCAService* e, portanto, exposto como um serviço *Web.* Esse método recebe como parâmetro uma lista de inteiros correspondentes a *tax ids* e retorna um objeto da classe *LCAEntryTO* contendo informações sobre o LCA calculado para os nodos associados aos *tax ids*. *LCAEntryTO* é uma versão da classe *TaxonEntry* adaptada ao protocolo SOAP (essa implementação segue o padrão de projetos *Transfer Object) [28]*. Internamente, *lca* chama o método *lowestCommonAncestor* do módulo LCA para calcular o LCA desejado.

A disponibilização do método LCA no paradigma REST é realizada pela classe *LCAController*, localizada no pacote *controller* do diretório da aplicação (ver Figura 5). O método *show* dessa classe lê os parâmetros passados pelo usuário na linha de comando do *browser* (ou em outros tipos de chamada HTTP). No caso do LCA, os *tax ids* vem separados pelo caractere "+" e são transformados em uma lista de inteiros. O método *lca* de *LCAService* é então chamado para o cálculo do LCA. O objeto da classe *LCAEntryTO* retornado é então convertido para o formato XML ou TSV, de acordo com a preferência do usuário. O texto resultante é retornado para o usuário.

O *Web service Taxonomy* também está disponível no paradigma REST e no protocolo SOAP. Sua construção é muito semelhante àquela do *Web service* LCA. Foi criada uma classe denominada *TaxonomyService* no diretório *services* da estrutura da aplicação, a qual possui o método *getAllChildrenGenomeTaxIds.* Esse método recebe como parâmetro um inteiro representando um *tax id* e retorna todos os descendentes folha do nodo associado a ele. A classe dos objetos da lista de retorno é denominada *GenomeTaxonInfo* e contém informações sobre os *taxa.* Internamente, *getAllChildrenGenomeTaxIds* chama o método *findAllDescendants* do módulo *TaxonomyTree.*

A disponibilização do método *getAllChildrenGenomeTaxIds* pelo paradigma REST segue a mesma metodologia que LCA. Nesse caso, a classe criada é denominada *TaxonomyController*.

A Figura 5 mostra a estrutura da aplicação *Grails* gerada. A premissa "código por convenção" da plataforma *Grails* permite que o *framework* "entenda" a função da cada arquivo e atributo simplesmente pelos seus nomes e/ou localização. Desse modo, não se torna necessária a criação de arquivos de configuração.

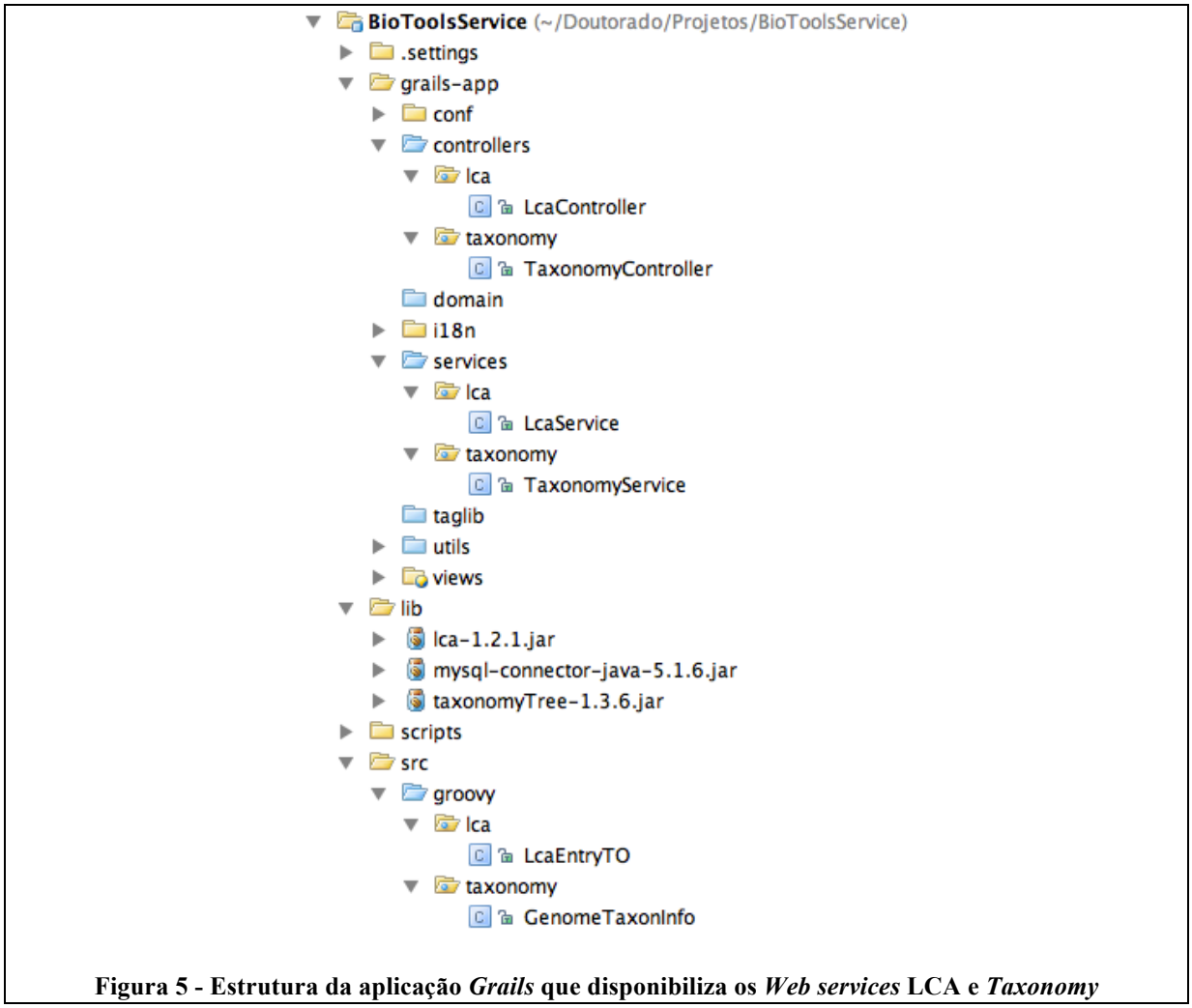


Figura 5 - Estrutura da aplicação *Grails* que disponibiliza os *Web services* LCA e *Taxonomy*

3.6 Implementação do módulo *UekoLCA*

O módulo *UekoLCA* tem a função de calcular o LCA de um grupo KO ou UEKO. O módulo é implementado na linguagem *Groovy* e é composto por uma classe principal e dois *scripts*. A classe principal, cujo nome é também *UekoLCA*, possui o método *findLCAForKO,* o qual tem a função de encontrar o LCA para um determinado grupo KO ou UEKO.

O módulo *UekoLCA* é dependente dos módulos *TaxonomyTree* e LCA e também utiliza o *framework* Spring. Assim que *UekoLCA* é executado, o módulo *TaxonomyTree* carrega a árvore taxonômica em memória para que o método *findLCAForKO* possa ser executado.

Para obter os grupos KOs e os *tax ids* associados a eles, o módulo faz consultas à tabela UEKO, a qual possui a estrutura demonstrada na Figura 6. A tabela foi construída no projeto de doutorado do pesquisador Gabriel R. Fernandes[17], e cedida pelo autor.

UekoLCA pode ser utilizado como um componente de outro módulo ou pode ser executado via linha de comando por meio de seus *scripts.* O *script UekoLCAFileRunner* objetiva encontrar o LCA de uma lista de grupos KO (identificados por seu *K identifier)* passados em um arquivo de entrada.

O *script UekoLCARunner* calcula o LCA de todos os grupos KO associados a uma determinada espécie e grava o resultado em uma tabela de banco de dados. Para isso, recebe como entrada um *tax id* e o nome da tabela de banco de dados destino. O *script* seleciona então na tabela UEKO os grupos KO distintos que possuam ao menos uma proteína da espécie identificada pelo *tax id* passado por parâmetro.

A definição entre as bases KO ou UEKO para a pesquisa também é um parâmetro de entrada. A diferença na consulta é que para KO, os filtros de proteína e espécie são realizados nas colunas *origin* e *tx_o,* respectivamente, enquanto que para UEKO são realizados nas colunas *enrich* e *tx_e.* A coluna *enrich* foi criada pelo autor desse banco de dados para armazenar os identificadores das proteínas que foram recrutadas para um grupo KO pelo algoritmo de enriquecimento do grupo.

De posse da lista de grupos KO distintos para a espécie, pode-se então calcular o LCA para cada um deles. O método *findLCAForKO* recebe um grupo KO como parâmetro e retorna um objeto da classe *TaxonEntry* contendo informações sobre o LCA. Para isso, encontra todos os distintos *tax ids* na tabela UEKO que estão associados ao grupo KO. A consulta SQL a seguir é executada:

```
SELECT DISTINCT <coluna_txid>
FROM ueko
WHERE
<coluna_ko> = <k_number>
```

na qual <coluna_txid> pode ser "tx_o", ou "tx_e" para consulta em KO ou UEKO, respectivamente, <coluna_ko> pode ser "origin" para KO ou "enrich" para UEKO e <k_number> é o identificador do grupo KO a pesquisar.

O método *lowestCommonAncestor* do módulo LCA é então executado para o grupo de *tax ids* obtido, e retorna um objeto da classe *TaxonEntry* com as informações do LCA.

O procedimento acima é realizado para cada grupo KO encontrado para a espécie pesquisada e os dados dos LCAs retornados são gravados na tabela destino do banco de dados. A Figura 6 ilustra a estrutura da tabela *hsa_ueko_lca*, que foi gerada para *Homo sapiens (tax id 9606)*. Observa-se que cada registro da tabela grava informações sobre o *tax id,* nome, *rank* e profundidade (*level)* do LCA de cada grupo KO associado à espécie.

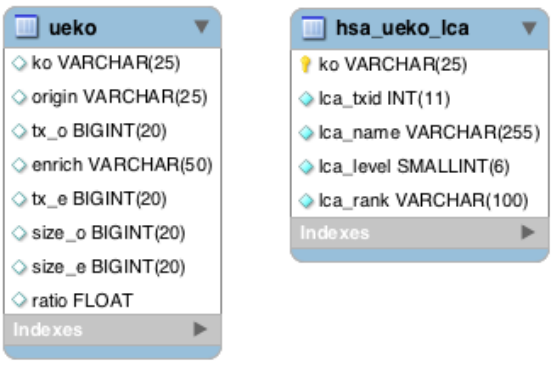


Figura 6 - Tabelas *ueko* e *hsa_ueko_lca*

3.7 Implementação do *script* KGMLtoDB

O *script* KGMLtoDB tem a função de inserir no banco de dados local as informações de arquivos KGML (*KEGG Markup Language*) disponibilizados pela base KEGG. Esses arquivos estão em formato XML e seguem regras específicas para a descrição de vias metabólicas através de seus elementos e relacionamentos. Na data em que este trabalho foi desenvolvido, a KEGG disponibilizava esses arquivos gratuitamente em seu servidor FTP. Entretanto, na data que este texto foi escrito, o acesso ao FTP passou a ser restrito. Como alternativa, os arquivos KGML podem ser obtidos no *Web service* disponibilizado pela KEGG [31].

O *script* KGMLtoDB realiza, portanto, o *parsing* de arquivos KGML e popula tabelas no banco de dados local, no qual se pode extrair de forma relevante e eficiente as informações contidas nesses arquivos.

Os arquivos KGML possuem *tags* específicas que identificam os elementos (*tag* <*entry*>, os relacionamentos (*tag* <*relation*>) e as reações (*tag* <reaction>), presentes em uma via metabólica. Para este trabalho, entretanto, interessam apenas os elementos e os relacionamentos entre eles. Por isso, o *parsing* realizado pelo *script* KGMLtoDB extrai primeiramente os elementos e os insere no banco de dados. Em seguida, extrai os relacionamento entre eles e também efetua inserção.

A Figura 7 ilustra um exemplo do código de um arquivo KGML com as três possíveis *tags entry, relation* e *reaction (*apenas as partes relevantes do código são exibidas).

```
<?xml version="1.0"?>
<!DOCTYPE pathway SYSTEM "http://www.genome.jp/kegg/xml/KGML_v0.7.1_.dtd">
<pathway name="path:hsa00010" org="hsa" number="00010"
         title="Glycolysis / Gluconeogenesis"
         image="http://www.genome.jp/kegg/pathway/hsa/hsa00010.png"
         link="http://www.genome.jp/kegg-bin/show_pathway?hsa00010">
    <entry id="13" name="hsa:226 hsa:229 hsa:230" type="gene" reaction="rn:R01070"
        link="http://www.kegg.jp/dbget-bin/www_bget?hsa:226+hsa:229+hsa:230">
    </entry>
    <entry id="37" name="hsa:217 hsa:219 hsa:223 hsa:224 hsa:501" type="gene" reaction="rn:R00710"
        link="http://www.kegg.jp/dbget-bin/www_bget?hsa:217+hsa:219+hsa:223+hsa:224+hsa:501">
    </entry>
    .
    .
    .
    <relation entry1="13" entry2="37" type="ECrel">
        <subtype name="compound" value="87"/>
    </relation>
    .
    .
    .
     <reaction id="37" name="rn:R00710" type="reversible">
        <substrate id="102" name="cpd:C00084"/>
        <product id="40" name="cpd:C00033"/>
    </reaction>
</pathway>
```

Figura 7 - Exemplo de um arquivo descritor de vias metabólicas KGML

Três tabelas são geradas no início da execução do *script: path_entry, entry_relationship* e *entry_elements.* A Figura 8 apresenta os modelos físicos de cada uma das tabelas.

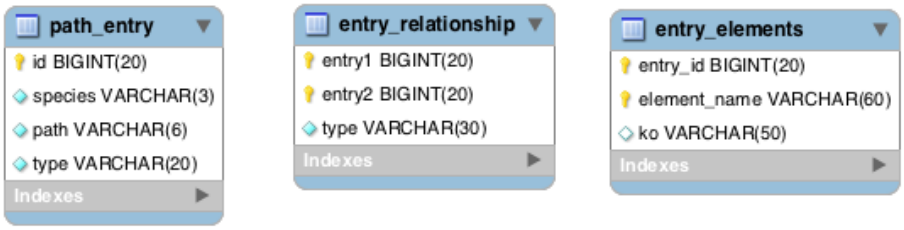


Figura 8 - Modelos físicos das tabelas *path_entry, entry_relationship* e *entry_elements*

A tabela *path_entry* armazena os elementos de uma via metabólica. A coluna *type* identifica o tipo de elemento, que pode ser gene ou outros compostos e substâncias. O código de três letras da espécie que possui o elemento (ex: *hsa)* é armazenado na coluna *species.* A coluna *path* identifica a via metabólica a qual o elemento está associado. A coluna *id* é um identificador único para o elemento (gerado automaticamente com a inserção).

A tabela *entry_relationships* guarda relacionamentos entre elementos. As colunas *entry1* e *entry2* compõe a chave dessa tabela. Ambas são chaves estrangeiras para a coluna *id* da tabela *path_entry* e armazenam os dois elementos que fazem parte do relacionamento. A coluna *type* denota o tipo de relacionamento (ex: enzima-composto, composto-composto, etc.).

A tabela *entry_elements* foi criada porque muitas vezes um único elemento está associado a mais de um gene ou grupo KO. Essa tabela armazena todos os possíveis nomes de um elemento. A coluna *entry_id* é uma chave estrangeira para a coluna *id* da tabela *path_entry*. O nome do elemento é armazenado na coluna *element_name.* A coluna *ko* é preenchida com o grupo KO associado do qual o nome do elemento faz parte.

A lógica do *script* KGMLtoDB consiste no *parsing* de todos os arquivos KGML disponíveis para *Homo sapiens* e armazenamento dos dados dos mesmos nas tabelas criadas. Para isso, utiliza-se a classe *XmlParser,* presente na biblioteca *Groovy.* Essa biblioteca permite a rápida consulta aos valores dos elementos e atributos de um arquivo no formato XML por meio da linguagem *XPath* (XML *Path Language*) [32].

Para cada arquivo KGML, todas as suas *tags* <*entry*> são lidas e suas informações gravadas como um novo registro na tabela *path_entry.* A tabela *entry_elements* também é preenchida com todos os possíveis nomes para o elemento. Depois disso, as *tags* <*relation*>são lidas para que os relacionamentos entre os elementos sejam armazenados na tabela *entry_relationship.* Ao fim, a coluna *ko* da tabela *entry_elements* é populada com os identificadores dos possíveis grupos KO associados a cada nome de elemento. Para isso, faz-se uso de um arquivo disponibilizado no servidor da KEGG que associa nomes de genes a grupos KO.

3.8 Construção das tabelas *path_subgraph* e *subgraph_element*

As vias metabólicas foram se compondo ao longo da evolução das espécies. Durante cada clado taxonômico, vias menores foram se agregando umas nas outras até que a via

chegasse ao estágio em que são hoje conhecidas. Neste trabalho, essas vias menores são denominadas subvias, pois representam uma fração da via metabólica completa.

Os modelos físicos das tabelas *path_subgraph* e *subgraph_element* são apresentados na Figura 9.

Cada registro da tabela *path_subgraph* identifica um estágio na evolução da via metabólica, o qual foi denominado EPV (espécie-profundidade-via). A profundidade (coluna *level_acc)* é representada por um inteiro que indica a distância do clado associado ao estágio até a raiz da árvore taxonômica. A coluna *level_name* armazena o nome do clado associado ao estágio. A tabela também armazena informações sobre o número de subvias que já existiam no estágio representado (coluna *num_clusters)* e também o número total de genes já existentes (coluna *num_entries)*. A coluna *num_clusters_last_level* armazena o número de subvias que a via possui no clado folha (isso é, na atualidade).

A tabela *subgraph_element* armazena os elementos das subvias. Todos os genes representados nessa tabela associados a um mesmo EPV estão presentes no mesmo estágio evolucionário da via. A coluna *cluster_seq* tem a função de discriminar a subvia que o gene pertencia, fornecendo um número sequencial para cada uma delas.

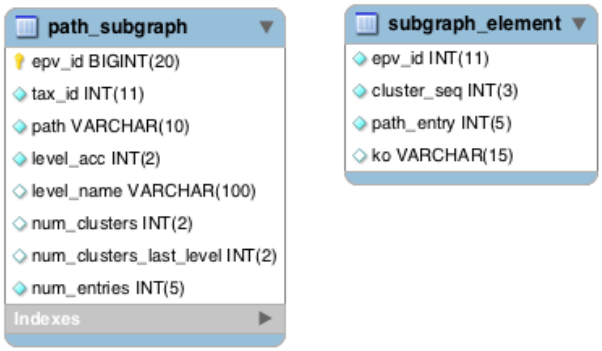


Figura 9 - Modelos físicos das tabelas *path_subgraph* e *subgraph_element*

Um *script* denominado *PathwaySubgraphs* foi construído na linguagem *Groovy* com o objetivo de construir e popular as tabelas *path_subgraph* e *subgraph_element.* Para a formação das subvias, o *script* utiliza o algoritmo *Single Linkage.*

O algoritmo *Single Linkage* é utilizado para criação de *clusters* a partir de um conjunto de relacionamentos e é utilizado nesta implementação para a determinação das subvias. Em um *cluster* formado por *Single Linkage* todo membro tem relacionamento com pelo menos

outro membro do *cluster*. Por exemplo, se um elemento A é relacionado a B, e B é relacionado a C, então A, B e C formam um *cluster.*

O pseudo-código a seguir descreve a lógica utilizada para que as tabelas sejam populadas:

1. Para cada via metabólica distinta presente na tabela *path_entry*:
    a. Armazena em memória todos os elementos associados ao nome e *level* de seu LCA
    b. Armazena todos os relacionamentos entre elementos
    c. Para cada profundidade i, a partir da 0:
        i. Encontra o todos os relacionamentos em que ambos elementos surgiram antes ou junto do clado de profundidade i
        ii. Executa *Single Linkage* para verificar o número de subvias presentes.
        iii. Insere EPV na tabela *pathway_subgraph*
        iv. Insere os elementos de cada subvia na tabela *subgraph_element* associando-os ao EPV criado

Ao final da execução, as tabelas estarão populadas e será então possível aferir o estágio da via (número de subvias, elementos das subvias, etc.) em qualquer clado da evolução da espécie que a possui.

3.9 Passo-a-passo do algoritmo de LCA Múltiplo

O programa que implementa o algoritmo de LCA Múltiplo foi criado na linguagem de programação *Groovy* e pode ser executado por linha de comando em sistemas que possuam a máquina virtual Java. O *script* aceita como entrada um arquivo de configuração, no qual é possível definir, entre outros ajustes, o banco de dados a utilizar. Além disso, nesse arquivo são informados os nomes dos arquivos contendo as listas de organismos que servirão como parâmetros de entrada.

3.9.1 *Lista positiva e lista negativa*

O algoritmo de LCA Múltiplo realiza em todas as suas etapas comparações na árvore taxonômica do NCBI. Portanto, a etapa inicial de sua execução é o carregamento da mesma em memória, exatamente da forma descrita na seção 3.2 deste trabalho. O programa recebe como entrada uma lista de organismos positivos (que possuem o gene), uma lista de organismos negativos (que não possuem o gene) e uma lista de nodos que cujos descendentes

não devem ser considerados na execução do algoritmo (nodos de exclusão). Cada lista deve conter um *tax id* em cada linha. Logo após o carregamento da árvore em memória, os nodos positivos e negativos são localizados, e aqueles que fazem parte da lista de exclusão são removidos.

3.9.2 *Construção da LCA Tree*

A primeira etapa do algoritmo de LCA múltiplo é a construção da *LCA Tree.* O objetivo dessa árvore é servir de base para a posterior detecção dos múltiplos de LCAs, por meio do atravessamento da mesma utilizando critérios específicos. As técnicas utilizadas para a detecção dos múltiplos LCAs são descritas na seção 3.9.3. Nesta seção são detalhados os métodos utilizados para a construção da *LCA Tree.*

A estrutura da *LCA Tree* é similar à utilizada para a árvore taxonômica, descrita na seção 3.2. São também utilizados objetos da classe *DefaultMutableTreeNode* para representar os nodos da árvore. A principal diferença entre as duas árvores está nos tipos de *userObject* associados a cada nodo. Enquanto a árvore taxonômica utiliza objetos do tipo *TaxonEntry*, que contêm informações relativas a um táxon, a *LCA Tree* associa objetos do tipo *LCAInfo* a seus nodos. Objetos da classe *LCAInfo*, além de possuírem dados taxonômicos, também contém informações adicionais que servem como base para posterior classificação de um nodo como um LCA.

O primeiro passo para a construção de uma *LCA Tree* é a determinação do LCA Base. Esse passo nada mais é do que a utilização do algoritmo de LCA simples sobre a lista de positivos. O LCA Base é o mais remoto ancestral comum de todos os organismos que possuem o gene estudado, portanto todos os LCAs alternativos encontrados serão certamente seus descendentes. Assim, o LCA Base constitui a raiz da *LCA Tree*. É importante notar que o LCA Base poderá ou não estar incluído na lista final de LCAs múltiplos encontrados pelo algoritmo. Isso acontecerá se o nodo raiz atender os critérios para a caracterização de um nodo como LCA, os quais são listados posteriormente neste texto.

Depois da criação do nodo raiz da *LCA Tree*, se inicia o processo de inserção dos demais nodos. Para isso, árvore taxonômica original é atravessada recursivamente, a partir do LCA Base e então passando em todos os seus descendentes. Para cada nodo da árvore taxonômica atravessado, o algoritmo poderá manter ou não um nodo correspondente na *LCA Tree*. Um nodo é mantido na *LCA Tree* somente quando:

a. representa um dos organismos da lista de positivos ou da lista de negativos, ou,
b. é ancestral de organismos positivos ou negativos e possui mais de um descendente direto na árvore (nodos que possuem apenas um descendente são substituídos por esse descendente).

Os nodos que se qualificam para estarem representados na *LCA Tree* são incluídos na árvore com informações fundamentais para a sua posterior caracterização como um dos LCAs. Essas informações são guardadas em objetos da classe *LCAInfo*. A principal informação de um objeto da classe *LCAInfo* é o seu tipo, o qual pode assumir os seguintes valores:

- GENOMA_POSITIVO: quando corresponde a um dos organismos da lista de positivos (geralmente são nodos-folha da árvore taxonômica).

- GENOMA_NEGATIVO: quando corresponde a um dos organismos da lista de negativos (geralmente são nodos-folha da árvore taxonômica).

- MISTO: quando possui no mínimo um descendente direto positivo (ou genoma positivo) e no mínimo um descendente direto negativo.

- POSITIVO: quando não é um nodo misto e possui dois ou mais descendentes diretos viáveis. São aqui denominados viáveis aqueles nodos que possuem o gene ou têm a capacidade de transferir o gene para seus descendentes, isso é, os positivos, genoma positivos e mistos.

- NEGATIVO: quando possui apenas descendentes diretos do tipo negativo ou no máximo um descendente direto viável.

Há uma exceção na lógica acima que ocorre quando o nodo misto possui apenas um genoma descendente negativo. Nesse caso, o nodo é considerado positivo e o genoma descendente negativo encontrado é considerado como alvo de perda gênica ou falso negativo. A lógica é realizada dessa forma porque a presença de apenas um genoma descendente

negativo é considerada prova muito fraca para determinar a ausência de um gene em um ramo da árvore.

Além do tipo, também são adicionadas outras informações a um objeto *LCAInfo*. Essas informações estarão presentes na *Mini LCA Tree* final, e poderão ser úteis para a diferenciação de surgimentos *de novo* de transferências horizontais. As informações adicionais incluídas em um objeto *LCAInfo* são:

- Número total de descendentes genoma positivo e número total de descendentes genoma negativo.
- Número total de nodos-folha com classificação desconhecida.
- Nodos de prova positivos e negativos.

Os nodos de prova são usados na construção da *Mini LCA Tree*, como será visto posteriormente neste texto. Se um nodo foi classificado como positivo, seus nodos de prova associados são dois descendentes genoma positivo. É necessário também que o caminho até cada descendente na árvore seja por meio de um filho direto diferente. Do mesmo modo, para o caso de um nodo classificado como negativo, são associados dois descendentes genoma negativo e, para o caso de nodos mistos, são associados um nodo genoma positivo e um genoma negativo. A utilização de nodos de prova na representação visual da *Mini LCA Tree* ilustra com maior clareza os motivos de um clado ter sido escolhido LCA.

3.9.3 *Detecção de múltiplos LCAs*

Depois da *LCA Tree* pronta, o próximo passo do algoritmo de LCA Múltiplo é a detecção dos LCAs. Os clados da *LCA Tree* são criados com as informações necessárias para a aplicação dos critérios que os classificam como LCA ou não. A rotina de detecção de LCAs faz o atravessamento recursivo da *LCA Tree*, começando pelo nodo raiz e atravessando todos os seus nodos. Para cada nodo atravessado, a rotina utiliza critérios pré-determinados para a sua caracterização como um LCA. Os critérios são construídos com base na posição do nodo na árvore, no seu tipo e nos tipos do nodo pai e dos nodos filhos. Dessa forma, um clado é caracterizado como LCA quando seu nodo atende pelo menos um dos três critérios a seguir:

a) O nodo é positivo e possui pai negativo
b) O nodo é o único filho positivo de pai misto

c) O nodo é misto, possui pai negativo e mais de um filho viável

Quando um nodo atravessado atende um dos critérios acima, ele é adicionado para o conjunto de múltiplos LCAs que representam as prováveis origens do gene em análise. Além da indicação do clado de origem, cada elemento do conjunto traz informações adicionais como o número de genomas positivos e negativos sob o LCA. Esses dados são extremamente úteis na determinação da qualidade do resultado e das possíveis origens causadas por transferências laterais.

Juntamente com o conjunto de LCAs, o sistema também produz o conjunto de organismos que sofreram possíveis perdas gênicas. Os elementos desse conjunto são determinados pela lógica descrita acima neste texto (únicos descendentes negativos de nodos mistos).

O conjunto de LCAs e o conjunto de organismos com perdas gênicas são os dois primeiros produtos do algoritmo de LCA Múltiplo. O terceiro produto do algoritmo é a *Mini LCA Tree*, uma versão enxuta da *LCA Tree* que facilita a visualização da árvore em softwares gráficos. A lógica para a construção da *Mini LCA Tree* é descrita na próxima seção.

3.9.4 *Construção da Mini LCA Tree*

A *LCA Tree* construída na primeira etapa do algoritmo de LCA Múltiplo pode também ser utilizada para visualização gráfica e melhor entendimento dos resultados. Porém, por ser uma árvore auxiliar gerada apenas como base para a detecção dos LCAs, a *LCA Tree* poderá conter de centenas a milhares de nodos. Essa complexidade dificulta a utilização da mesma para a representação visual dos resultados, uma vez que seria extremamente grande e confusa. Para resolver esse problema, o último passo do algoritmo gera uma versão compacta da *LCA Tree*, a qual foi denominada *Mini LCA Tree*.

A rotina de criação da *Mini LCA Tree* inclui dois passos. O primeiro é novamente o atravessamento recursivo da *LCA Tree*. O objetivo é a geração de uma nova árvore contendo apenas os nodos essenciais para a representação gráfica dos resultados. Um nodo é considerado essencial para exibição somente quando se enquadra em um dos critérios a seguir:

a) Representa um dos LCAs.
b) Algum de seus filhos diretos é LCA.

c) Representa um dos clados da lista *alwaysShow*. Essa lista contém nodos que sempre devem ser exibidos na representação gráfica da árvore de LCAs. Neste trabalho optou-se por sempre se exibir *Eukaryota*, *Bacteria* e *Archaea*, por serem clados de referência.

O segundo passo para a montagem da *Mini LCA Tree* é a adição dos nodos de prova. Os nodos de prova são úteis na representação gráfica de uma árvore de LCAs pois permitem que o pesquisador facilmente entenda o motivo que levou um nodo a ser qualificado como positivo, negativo ou misto. A adição de nodos de prova à *Mini LCA Tree* ocorre depois que os nodos essenciais já foram adicionados. A árvore é então atravessada partindo da raiz até as folhas. Para cada nodo percorrido, seu tipo é verificado e as provas são adicionadas à árvore com base nessa informação. É importante lembrar que o conjunto de nodos de prova de cada nodo da *LCA Tree* já havia sido incluído durante a criação da mesma, como descrito na seção anterior. Desse modo, o único trabalho da rotina de montagem da *Mini LCA Tree* é verificar o tipo do nodo e adicionar as provas à árvore de acordo com ele.

3.10 Desenvolvimento do *website* Genesis

O *website* Genesis provê rápido e fácil acesso a resultados pré-computados do LCA múltiplo, tanto em nível de genes individuais (representados por seus grupos de ortólogos KO) quando de espécies como um todo. Além disso, o *website* permite que o pesquisador execute análises de origem múltipla com listas de organismos customizadas. Para isso, o sistema executa o algoritmo de LCA múltiplo descrito na seção anterior em tempo real. Nesta seção são descritos os materiais e métodos utilizados para a construção do *website* Genesis. Os resultados podem ser encontrados no capítulo 5.

Assim como os demais programas desenvolvidos neste trabalho, o *website* Genesis também foi construído com o *framework Grails* versão 2.3.2 [21]. Esse *framework* propicia grande aumento na produtividade por meio de vários recursos que evitam tarefas repetitivas de codificação. A linguagem de programação utilizada em *Grails* é *Groovy*.

Para a montagem das telas foi utilizado o *framework* de *front-end Bootstrap* versão 3.0.2 [33]. *Bootstrap* permite a construção de *websites* responsivos com grande facilidade, o que possibilitou que Genesis se tornasse acessível tanto de *desktops* quanto de *smartphones* e *tablets*.

Além disso, foi usada a biblioteca *jQuery* versão 1.10.2 [34], a qual facilita a manipulação de elementos e eventos da linguagem HTML.

O gerenciador banco de dados utilizado foi o *MySQL Community Edition* [22] (versão 5.1.59).

O *website* Genesis possui três ferramentas para inferência da múltipla origem cladística de genes. A primeira, denominada *KO origin,* permite que o pesquisador explore as múltiplas origens de um gene específico. A segunda, chamada de *Species origin,* realiza a análise de ancestralidade gênica em um âmbito de espécies. A última ferramenta permite que o pesquisador realize a análise de origem múltipla de genes em um conjunto personalizado de organismos de entrada. Nas próximas seções são descritos os métodos utilizados para a construção de cada uma dessas ferramentas.

Todo o código escrito para *Genesis* está disponível em código aberto no endereço https://github.com/hvmelo/Genesis.

3.10.1 *KO Origin*

A ferramenta de busca pelas múltiplas origens de um gene recebe como entrada do usuário um identificador de grupo de ortólogos KO e deve retornar os LCAs resultantes do algoritmo de LCA múltiplo.

Para que a resposta seja rápida, os LCAs Múltiplos foram pré-computados e armazenados em um banco de dados do servidor. Assim, o algoritmo de LCA Múltiplo foi executado em todos os grupos de ortólogos KO da base UEKO. Para isso, foi necessário encontrar os *tax ids* de todos organismos de um grupo. Esse dado é facilmente extraído do bando de dados UEKO, como é visto na seção 3.6. Essa lista de *tax ids* constitui a lista positiva que deve ser fornecida ao algoritmo de LCA múltiplo.

A lista negativa é obtida dos genomas completos que não possuem o gene estudado. Para o *website Genesis,* a lista negativa é obtida do site de genomas completos do NCBI acessado pelo endereço http://www.ncbi.nlm.nih.gov/genome.

Com a execução do algoritmo de LCA múltiplo em cada grupo de ortólogos, uma nova tabela chamada *multi_lca* foi criada e disponibilizada para acesso via servidor *Web*. Esta tabela possui as seguintes colunas:

ko: Identificador do grupo de ortólogos KO.
ko_desc: Descrição do gene representado pelo grupo KO.
lca_txid: NCBI Taxonomy ID do LCA.
lca_name: Nome científico do clado do LCA.
lca_level: Profundidade do LCA (distância até a raiz).

lca_rank: Rank do LCA (classe, família, reino, etc.).
multiple: Informa se o LCA é múltiplo ou simples.
size: O número de genomas positivos sobre o LCA.

É importante notar que cada grupo KO pode ter um ou mais registros nessa tabela. Portanto, quando o usuário de Genesis requisita os múltiplos LCAs de um grupo KO, eles são rapidamente recuperados da tabela *multi_lca* em uma consulta simples do tipo:

```
SELECT *
FROM multi_lca
WHERE ko = 'K00226'
```

3.10.2 *Species Origin*

Na ferramenta de análise da origem múltipla de genes em espécies disponibilizada pelo *website* Genesis, o usuário entra com o *tax id* de um dada espécie e o sistema retorna com uma série de relatórios relativo às origens de seus genes.

Para chegar aos resultados acima, o sistema executa uma série de rotinas de sumarização utilizando a tabela pré-computada *multi_lca*. Essa tabela contém os múltiplos LCAs computados para cada KO, como descrito na seção anterior. Entretanto, dessa vez, o taxonomy id da espécie pesquisada é utilizado na consulta, como no exemplo abaixo:

```
SELECT DISTINCT M.KO, M.KO_DESC, M.LCA_NAME, M.LCA_TXID,
M.LCA_LEVEL
FROM MULTI_LCA M INNER JOIN UEKO U ON M.KO = U.KO
WHERE U.TAXID = 9606
```

A consulta acima retorna todos os múltiplos LCAs de todos os grupos de ortólogos KO da espécie humana. Os resultados são então agrupados por grupo KO. Em seguida, é necessário isolar os grupos KO que aparecem por suposta transferência lateral daqueles que realmente se originaram em um clado ancestral da espécie. O LCA escolhido para o gráfico gerado é aquele ancestral mais próximo do nodo da espécie investigada.

3.10.3 *Multi LCA Tree*

Diferentemente das outras funcionalidades do *website* Genesis, a *Multi LCA Tree* não é pré-computada, e sim, gerada dinamicamente a partir da entrada de dados do usuário. Por meio da interface do *website,* o usuário entra com uma lista de *tax ids* positivos (que possuem o gene em estudo) e uma lista de *tax ids* negativos (que não possuem o gene em estudo). A partir dessas duas listas, o sistema executa o *script* de LCA Múltiplo em tempo real para a construção de uma *Mini LCA Tree.*

Como o LCA é calculado em tempo real, a árvore taxonômica é carregada em memória assim que o servidor é iniciado. Para isso, os *scripts* de carregamento da árvore e de LCA foram incluídos na pasta *src/groovy* do diretório da aplicação. Também foi incluído o *script MultipleLCA,* responsável pela execução do algoritmo de LCA Múltiplo. A partir disso foram criados *Spring beans* no arquivo *BootStrap.groovy*, o qual é executado na inicialização do servidor. Essa estratégia é muito similar ao padrão *Singleton*, isto é, permite que apenas uma instância de cada classe seja criada e compartilhada entre os objetos que dela necessitam. Os *beans* foram injetados nos controladores e serviços que deles necessitam.

O controlador *MultiLCAController* possui o método *multiLCATree()* responsável por receber a entrada de dados do usuário e disponibilizar os resultados. A entrada de dados é lida pela variável *params.* Essa variável é um *map* com três pares chave/valor preenchidos no GSP (camada de visualização) responsável pela criação da tela a partir de seus campos. Eles são:

- *params.positiveList*: Carrega a lista de *Taxonomy Ids* de genomas positivos do GSP para o controlador.
- *params.negativeList*: Carrega a lista de *Taxonomy Ids* de genomas negativos do GSP para o controlador.
- *params.loadNegatives*: Variável booleana que informa se todos os genomas não listados entre os positivos devem ser considerados negativos.

A classe de serviço *GeneOriginService* possui o método *findMultipleLCAs*, responsável por encontrar os múltiplos LCAs a partir de uma lista de genomas positivos e uma lista de genomas negativos. Esse método é chamado a partir do controlador *MultiLCAController* passando-se as listas entradas pelo usuário. Caso *loadNegatives* seja positivo, a lista de negativos é calculada subtraindo-se os genomas positivos da lista total de genomas negativos do sistema. A classe *GeneOriginService* possui uma instância *Singleton* do *bean MultipleLCA*, o qual possui o *script* de execução do algoritmo de LCA Múltiplo. *MultipleLCA* possui o método *findMultipleLCAs* que recebe como entrada quatro parâmetros.

Além da lista de positivos e negativos, também deve ser passado para esse método os *taxIds* que não devem ser considerados na execução e os *taxIds* dos clados que sempre devem aparecer nos resultados. Para o caso de Genesis, esses são parâmetros fixos. De forma geral, se exclui dos cálculos todos os vírus, viroides e seus descendentes, e sempre se exibe os três maiores domínios da árvore da vida: *Bacteria, Eukaryota* e *Archaea.*

O algoritmo de LCA Múltiplo é então executado em tempo real. A lógica do algoritmo é exatamente a mesma descrita na seção 3.9.2. O método apresenta como saída um mapa com três duplas chave/valor:

- *lcaSet:* O conjunto de LCAs encontrados, armazenados em objetos do tipo *LCAInfo.*
- *geneLosses:* O conjunto de possíveis perdas gênicas detectadas, também armazenadas em objetos do tipo *LCAInfo.*
- *miniLCATree:* O nodo raiz da *Mini LCA Tree* construída pelo algoritmo.

Os dois primeiros valores são utilizados no GSP de resposta para a listagem dos LCAs encontrados e das possíveis perdas gênicas, respectivamente. O nodo raiz da *Mini LCA Tree* é utilizado para a representação visual da árvore. Para isso foi utilizada a biblioteca Protovis [35]. A árvore é atravessada e transformada em um mapa hierárquico contendo informações sobre cada nodo em cada um de seus elementos. Esse mapa é posteriormente transformado no formato JSON já no lado do cliente, em código *Javascript*. Utiliza-se o *layout Indent*, já existente na biblioteca Protovis para disposição da árvore como uma estrutura hierárquica indentada.

3.11 BOWS – *Bioinformatics Open Web Services*

BOWS é uma plataforma que intermedeia o acesso de programas de usuários a ferramentas de Bioinformática que exigem processamento de alto desempenho. O produto é disponibilizado como um arquivo *war* que deve ser instalado em um servidor com acesso a Web.

A plataforma BOWS foi construída na linguagem *Groovy*, com a utilização do *framework Grails*. BOWS basicamente se trata de um conjunto de *Web services* que fornecem acesso seguro a um banco de dados. São disponibilizados dois *endpoints:* um administrativo ou de *back-end*, o qual é utilizado pelos donos das ferramentas e outro de *front-end*, o qual é usado pelos usuários interessados nas ferramentas.

BOWS mantém um banco de dados único onde armazena as ferramentas registradas, os processos em andamento e os resultados. O banco de dados é constituído de cinco tabelas, como demonstrado na Figura 10. As tabelas são descritas a seguir:

- *application:* armazena as ferramentas registradas na plataforma BOWS. Essa tabela possui os seguintes campos:
  - *code:* guarda o código único da ferramenta (ex: *blast).*
  - *name:* o nome da aplicação.
  - *description:* uma descrição para a aplicação.
- *application_param:* armazena os parâmetros de cada ferramenta registrada. Possui os seguintes campos:
  - *application_id:* liga o parâmetro à ferramenta ao qual está associado.
  - *name:* o nome do parâmetro.
  - *default_value:* o valor padrão para o parâmetro (usado quando um outro valor não é enviado).
  - *description:* descrição do parâmetro.
  - *examples:* alguns exemplos de uso.
  - *mandatory:* informa se o parâmetro é obrigatório.
  - *matcher:* uma expressão regular que deve validar o valor do parâmetro.
  - *value_binary:* informa se o valor do parâmetro é binário.
- *process:* armazena os processos submetidos. Possui os seguintes campos:
  - *application_code:* o código da ferramenta ao qual o processo foi submetido.
  - *date_created:* a data de submissão do processo.
  - *error_description:* guarda a descrição do erro de execução, caso haja.
  - *status:* guarda o *status* atual do processo.
- *process_param:* guarda os parâmetros de um processo submetido. Possui os seguintes campos:
  - *name:* o nome do parâmetro do processo.
  - *process_id:* o identificador do processo relacionado ao parâmetro.
  - *value_binary:* informa se o valor do parâmetro é binário.
  - *binary_value:* o valor binário do parâmetro submetido, caso o seja.
  - *text_value:* o valor textual do parâmetro submetido, caso o seja.
- *result:* armazena os resultados de um processo. Possui os seguintes campos:

- *process_id:* o identificador do processo ao qual o resultado está relacionado.
- *type:* indica se o resultado é binário ou textual.
- *result_binary:* o resultado em formato binário (BLOB).
- *result_text:* o resultado em formato textual.
- *mimeType:* o tipo MIME do resultado.

Além dos campos citados acima, todas as tabelas também possuem os campos *id* e *version*, os quais são automaticamente inseridos pela camada de persistência da plataforma *Grails*. O primeiro representa a chave primária da tabela e o segundo controla a versão dos dados.

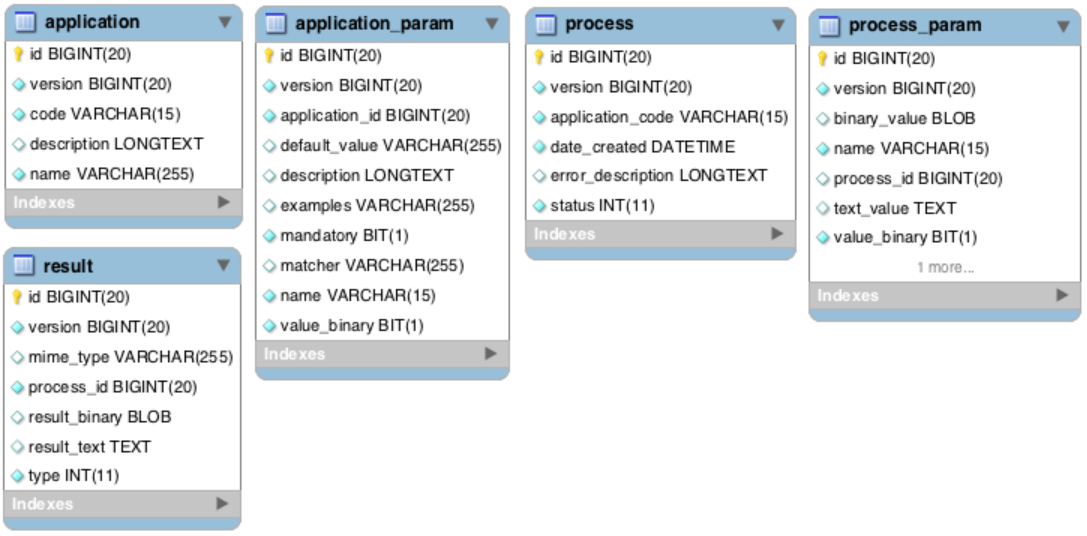


Figura 10 - Diagrama ER do banco de dados de BOWS

Cinco classes de domínio da plataforma acompanham as tabelas criadas. São elas: *Application, ApplicationParam, Process, ProcessParam* e *Result.* Cada uma das classes corresponde a uma tabela no banco de dados. Alterações nas instâncias dessas classes refletem automaticamente no banco de dados, por intermédio do *framework Grails*. Além disso, foram criadas duas enumerações: *DataType* e *ProcessStatus,* as quais correspondem respectivamente ao tipo de dados (binário ou textual) e ao *status* de um processo submetido (QUEUED, RUNNING, FINISHED e ERROR).

Foram também criadas classes de transporte cujo objetivo é transmitir dados através dos *Web services*. Essas classes são: *ApplicationParamConfig, DefaultResponse,*

ProcessDetailsResponse, ProcessResultParam, ProcessResultResponse, ProcessStatus Response, SubmissionParam e *SubmissionResponse.* Instâncias dessas classes carregam os dados correspondentes entre o cliente e o servidor.

Para a geração e exposição do métodos de *back-end* e *front-end* da plataforma BOWS foi utilizado o *plugin* CXF para *Grails* [36]. Por meio deste *plugin,* a simples declaração `static expose=['cxf']` faz com que todos os métodos de uma classe de serviços sejam expostos como serviços *Web.* Foram, portanto, criadas duas classes de serviço: *BOWSService* e *BOWSAdminService*. A primeira contém os métodos do serviço de *front-end* enquanto a segunda contém os métodos de *back-end.*

Além das classes citadas acima, foi criada uma classe para tratar as exceções da aplicação, chamada *BOWSException.* Essa classe carrega para o cliente os eventuais erros de validação ocorridos na chamada a algum dos métodos da plataforma.

4. FERRAMENTAS E SERVIÇOS TAXONÔMICOS

Os bancos de dados públicos de Bioinformática fornecem uma fonte valiosa de informações relativas aos genes presentes tanto em espécies existentes quanto extintas. Por meio dessas informações e do uso de ferramentas computacionais e de pesquisas na árvore taxonômica é possível estimar informações relevantes a respeito da ancestralidade de genes e de vias metabólicas.

Neste capítulo são apresentadas algumas ferramentas computacionais e *Web services* construídos para a mineração e manipulação de dados taxonômicos e os resultados biológicos que podem ser alcançados por meio dessas ferramentas.

Na seção 4.1 é apresentado *TaxSimple,* um *script* que "planifica" a árvore taxonômica do NCBI, permitindo que as consultas simples sejam realizadas rapidamente.

A seção 4.2 descreve o carregamento da árvore taxonômica do NCBI em memória. Essa etapa é fundamental para as demais ferramentas taxonômicas descritas neste texto, uma vez que viabiliza a execução de pesquisas taxonômicas extremamente complexas em milésimos de segundo.

O programa *GetAllChildren* é o primeiro resultado do carregamento da árvore taxonômica em memória. Por meio dele, o bioinformata é capaz de obter todos os nodos de um mesmo *rank*, descendentes de um determinado nodo pai. Esse programa é descrito na seção 4.3.

A seção 4.4 descreve o programa *LCAFinder,* cuja função é encontrar rapidamente o ancestral comum de dois ou mais nodos da árvore taxonômica do NCBI. Esse programa constitui a base de todos os demais estudos taxonômicos realizados neste trabalho, e consequentemente, dos resultados biológicos obtidos.

O programa *LCAFinder,* aliado às valiosas informações de grupos ortólogos contida no banco de dados *UEKO,* permitiu a criação do programa *GeneOrigin.* Esse programa calcula o ancestral comum de um grupo de genes ortólogos, permitindo inferir com boa precisão o clado de nascimento de um gene ancestral na árvore da vida. Curiosamente, os gráficos gerados sugerem um padrão de ondas de surgimento de genes. A seção 4.5 descreve o programa *GeneOrigin,* exibe e discute os diversos resultados biológicos encontrados.

A seção 4.6 mostra que aparentemente há padrões de surgimento de genes em uma espécie e exibe o gráfico da origem gênica em *Homo sapiens.*

A seção 4.7 mostra uma comparação realizada entre os gráficos gerados para surgimento de genes humanos com gráficos similares gerados para o surgimento de funções e processos biológicos. Verificou-se que o padrão de ondas também aparece nos gráficos.

Os gráficos de surgimento de genes foram também construídos para diversas outras espécies de reinos e clados distintos, como *D. melanogaster, A. thaliana* e *C. elegans*. Da mesma forma que em *H. sapiens*, apresentaram um padrão de ondas no surgimento de genes. Os resultados são exibidos na seção 4.8.

Por último, os resultados de origem gênica obtidos são extrapolados para vias metabólicas humanas na seção 4.9. Essa seção mostra o passo-a-passo de surgimento da via metabólica JAK-STAT.

4.1 *TaxSimple*

O NCBI, principal centro de biotecnologia dos Estados Unidos, possui um *website* no qual disponibiliza para os pesquisadores sua base de dados taxonômicos. Esta base, hoje umas das principais referências em taxonomia, é atualizada à medida que novas descobertas vão alterando a concepção da comunidade científica sobre o processo de especiação.

É possível visualizar e navegar na árvore taxonômica atualizada do NCBI em seu *website.* Além disso, o centro disponibiliza um servidor de FTP, a partir do qual é possível baixar os arquivos brutos contando os dados de suas tabelas. No entanto, o *website* não dá suporte a consultas taxonômicas mais complexas, como por exemplo, a busca de todos os descendentes de um determinado nodo que pertencem a um determinado *rank (*por exemplo, "liste todas as espécies da classe *Mammalia"*). Tampouco é trivial realizar esse tipo de consulta por meio do carregamento dos arquivos de dados brutos em um banco de dados local, uma vez que a natureza dos mesmos é recursiva, exigindo pré-processamento.

O programa *TaxSimple* foi criado para viabilizar este tipo de consulta e outras similares de uma forma simples e rápida. O programa gera uma tabela na qual é possível realizar uma única consulta SQL para se obter, por exemplo, todas as espécies de uma determinada família, ou todas as famílias de um determinado reino, o que de outro modo seria inviável devido à natureza recursiva dos dados. O programa *TaxSimple* "planifica" a tabela taxonômica do NCBI com a criação de colunas para os *tax ids* dos principais *ranks* das linhagens.

Todo organismo que possui sequencias depositadas em bancos de dados públicos tem sua linhagem representada na tabela. A primeira coluna (*genome_tax_id*) armazena o *tax id*

do genoma sequenciado (utiliza o *tax_id* que acompanha a sequencia depositada). Em alguns casos trata-se de uma cepa, sorotipo, subespécie ou espécie; assim, a informação *genome_tax_id* pode ser coincidente com a informação das colunas espécie/subespécie ou não, o que facilita a busca da classificação completa para as sequencias depositadas. As demais 17 colunas representam os *tax ids* que aparecem na linhagem da espécie no sentido folha à raiz da árvore. Entretanto, como o número de níveis na linhagem de cada espécie é variável, apenas os *taxa* que possuem os principais *ranks* são representados. Os *ranks* que possuem colunas na tabela são: super-reino, reino, filo, subfilo, superclasse, classe, subclasse, superordem, ordem, subordem, superfamília, família, subfamília, gênero, subgênero, espécie e subespécie. Os demais *ranks*, por serem pouco frequentes e seu uso pouco informativo, e o *rank* denominado "*no rank",* não são representados (*no rank* é uma categoria que não participa da hierarquia de *ranks,* aparecendo em *taxa* que não possuem uma categorização formal de *rank*).

Os detalhes sobre a implementação do programa que constrói a tabela *TaxSimple* são descritos na seção 3.3.

A Figura 11 ilustra os resultados de uma consulta à tabela gerada por *TaxSimple*. Na consulta, são obtidas as linhagens de cinco espécies: *Arabidopsis thaliana* (com *tax id* 3702), *Oryza sativa* (4530), *Drosophila melanogaster* (7227), *Homo sapiens* (9606) e *Mus musculus* (10090). O comando SQL utilizado na consulta é apresentado abaixo:

```
SELECT *
FROM tax_simple
WHERE genome_tax_id in (9606, 3702, 10090, 4530, 7227);
```

Pode-se observar que para alguns *ranks* não existe um táxon associado, como no caso do *rank* classe de *Arabidopsis thaliana.*

genome_tax_id	superkingdom	kingdom	phylum	subphylum	superclass	class	subclass	superorder
3702	2759	33090	35493	NULL	NULL	NULL	71275	NULL
4530	2759	33090	35493	NULL	NULL	4447	4734	NULL
7227	2759	33208	6656	NULL	6960	50557	33340	NULL
9606	2759	33208	7711	89593	7776	40674	NULL	314146
10090	2759	33208	7711	89593	7776	40674	NULL	314146

order	suborder	superfamily	family	subfamily	genus	subgenus	species	subspecies
3699	NULL	NULL	3700	NULL	3701	NULL	3702	NULL
38820	NULL	NULL	4479	147367	4527	NULL	4530	NULL
7147	7203	43746	7214	43845	7215	32341	7227	NULL
9443	376913	314295	9604	207598	9605	NULL	9606	NULL
9989	33553	NULL	10066	39107	10088	862507	10090	NULL

Figura 11 - Exemplo de consulta na tabela gerada por *TaxSimple*

A tabela gerada por *TaxSimple* permite uma diversidade de consultas que não seriam possíveis na estrutura recursiva das tabelas taxonômicas originais carregadas do servidor FTP do NCBI. Pode-se, por exemplo, obter todas as espécies do filo *Chordata (tax id 7711)* com apenas um simples comando SQL, demonstrado a seguir:

```
SELECT species FROM tax_simple WHERE phylum = 7711;
```

A tabela gerada por *TaxSimple* é constantemente atualizada e o *dump* da última versão produzida é distribuído em servidor aberto para a comunidade. O atual *link* para download pode ser encontrado na página de *Web services* do Laboratório de Biodados, disponível em http://biodados.icb.ufmg.br/services.

TaxSimple tem sido utilizada em vários outros projetos do Laboratório de Biodados [37]. A tabela é utilizada, por exemplo, quando se procura mostrar qual o relacionamento entre uma proteína da base de dados KO e a proteína recrutada pelo processo de enriquecimento da mesma, executado pelo algoritmo UEKO. A tabela é também usada para se limitar a busca de homólogos com o pacote *SeedServer*. Espera-se utilizá-la também para limitar a mineração de texto a determinados clados, como por exemplo, a classe a qual o milho pertence.

Uma das limitações de *TaxSimple,* contudo, é que as consultas ficam restritas apenas aos *ranks* representados na tabela. Existem *taxa* relevantes na linhagem humana, por exemplo, *Coelomata, Vertebrata* e *Euteleostomi,* que são cadastrados como *"no rank"* e por isso não aparecem na tabela gerada por *TaxSimple.* De fato, a linhagem humana possui 36 clados no total, e apenas 14 desses são representados (sendo que, nesse caso, *genome_tax_id* coincide com a espécie).

Portanto, apesar de muito útil para rápidas consultas, a tabela *TaxSimple* possui limitações que inviabilizam seu uso em certos projetos.

4.2 Carregamento da árvore taxonômica em memória

Para a realização de consultas complexas na árvore taxonômica sem as limitações da tabela *TaxSimple*, foi construído um modulo para carregamento completo da mesma em memória RAM. Existe um *overhead* inicial nesse procedimento que consiste na leitura dos registros de cada nodo da árvore no banco de dados e a subsequente construção da hierarquia em memória. Entretanto, após o carregamento, a árvore completa pode ser acessada de forma

extremamente rápida, mesmo para as consultas e cálculos mais complexos. Os detalhes da implementação do módulo são descritos na seção 3.2.

Com a árvore carregada em memória, foram implementados métodos auxiliares que realizam consultas complexas e fornecem informações evolutivas sobre *taxa* e hierarquias. Entre os métodos criados, pode-se citar:

- comparação de dois *taxa* a fim de se verificar se existe relação de ancestralidade entre eles.
- obtenção de uma sub-árvore com raiz em um dos nodos.
- obtenção de todos os nodos "primos" de um determinado nodo, isto é, descendentes de um mesmo dado clado (que responda, por exemplo, à requisição "quero todos os *taxa* que possuem a *mesma classe* que a *espécie* com *tax id* 9606").
- obtenção do táxon que é ancestral comum de um dado conjunto de *taxa*.

Essas são operações na árvore taxonômica que têm sido usadas no Laboratório de Biodados da UFMG [37] em diferentes projetos relacionados a taxonomia e evolução. Dada a utilidade dessas operações, as mesmas também foram disponibilizados na forma de *Web services* a fim de atender à comunidade científica geral. *Web services* [38] permitem o acesso remoto a serviços a partir de praticamente qualquer linguagem de programação, por isso constituem em uma estratégia universal e eficiente para disponibilização de ferramentas para a comunidade científica.

O *Web service* de taxonomia está disponível para acesso SOAP [39]. O endereço do WSDL é http://biodados.icb.ufmg.br:8080/BioToolsService/services/taxonomy?wsdl. No momento, apenas um método está disponível para acesso público, o qual é descrito na próxima seção.

4.3 *GetAllChildren*

O *Web service* de taxonomia atualmente disponibiliza o método denominado *getAllChildrenGenomeTaxIds*. Esse método recebe como parâmetro um dado *tax id*. O retorno é uma lista de *taxa* que representam clados com sequencias na base *Taxonomy* (isso é, são espécies atuais ou extintas das quais há sequencias disponíveis em bancos de dados públicos) que são descendentes do clado com o *tax id* passado por parâmetro. Os *taxa* retornados são encapsulados em objetos com os atributos "*taxId", "name", "rank"* e "*level",* representando

o *tax id,* o nome científico, o *rank* e a profundidade (distância do nodo à raiz da árvore) do táxon. Dessa forma, a subsequente manipulação dos dados a partir de qualquer linguagem de programação fica bastante simplificada. Além de fornecer o *genome_tax_id*, o serviço também retorna *species_tax_id*, que pode ser coincidente ou não com o primeiro. Isso é útil para os casos, por exemplo, em que as sequencias são obtidas de subespécies ou cepas. Nesse caso, pode ser de interesse verificar qual a espécie que engloba essas variantes.

O *Web service* de *GetAllChidren* está disponível em formato REST. A chamada deve ser realizada no seguinte formato:

```
http://merengue.icb.ufmg.br:8080/BioToolsService/taxonomy/getAllChildrenGenomeTaxIds?taxId=<taxId>
```

no qual <taxId> é o *taxonomy id* do organismo que se deseja obter os nodos-folha (ex: 9606).

O retorno é um arquivo no formato XML pré-formatado. O elemento raiz do arquivo de retorno é denominado *entries.* Este nodo contém uma lista de nodos do tipo *genomeTaxon,* os quais de fato representam um táxon. Cada elemento *genomeTaxon* possui um atributo *id* que representa o *tax id* do clado. Os elementos filhos de um *genomeTaxon* contém as outras informações relativas ao clado. Elas são: *name, rank, level, speciesId, nameId, rankId* e *levelId.*

A Figura 12 ilustra um exemplo de retorno do serviço *GetAllChidren.* O parâmetro de entrada, nesse exemplo, é o *taxonomy id* 9605, que representa o gênero *Homo.* Como é possível observar, o gênero *Homo* apresenta três nodos-folha: as duas subespécies de *Homo sapiens* - *Homo sapiens neanderthalensis* e *Homo sapiens ssp. Denisova* - e a espécie *Homo heidelbergensis,* que divergiu de *Homo sapiens.* Como *Homo sapiens* não constitui um nodo-folha, seu clado não é diretamente retornado. No entanto, suas subespécies apresentam o elemento *speciesId,* que denotam a espécie que representam. Nota-se, portanto, que *GetAllChildren* sempre retorna os nodos que representam as espécies, juntamente com as subespécies e cepas encontradas.

Figura 12 - Exemplo de retorno do serviço *GetAllChidren*

4.4 LCA – *Lowest Common Ancestor*

A base da maioria das ferramentas desenvolvidas neste trabalho é a determinação do LCA (*lowest common ancestor* ou mais recente ancestral comum) de um grupo de *taxa.* Todos os cálculos efetuados posteriormente para a análise da ancestralidade de genes e vias metabólicas se baseiam nessa funcionalidade.

O conceito de LCA vem da teoria dos grafos, e só é aplicável em árvores que possuem uma raiz. Em uma dada árvore T, o LCA (ou ancestral comum mais próximo) de dois nodos *a* e *b* é definido como o nodo mais próximo de T (isso é, mais próximo das folhas) que possui tanto *a* e *b* como descendentes. Vale ressaltar que um nodo é considerado descendente dele próprio.

A Figura 13 ilustra em uma árvore arbitrária com sete nodos enraizados pelo nodo 1. No quadro da esquerda são listados LCAs de diferentes subconjuntos de nodos. A mesma lógica se estende para a árvore taxonômica do NCBI carregada em memória. O LCA em uma árvore taxonômica denota o ancestral comum mais recente de dois ou mais clados.

Neste trabalho, o algoritmo de LCA foi desenvolvido como um módulo que opera sobre a árvore taxonômica em memória e retorna o táxon ancestral comum de uma lista de *tax ids* passados como parâmetro. O módulo, compartilhado pela maioria das ferramentas desenvolvidas neste trabalho, possui um único método denominado *lowestCommonAncestor,*

que recebe como parâmetro uma lista de *tax ids*. Quando executado, a árvore taxonômica é carregada em memória e os nodos correspondentes aos *tax* ids de entrada são encontrados. O LCA é calculado e retornado para o programa que o requisitou. Os materiais e os detalhes de implementação do módulo LCA são descritos na seção 3.4.

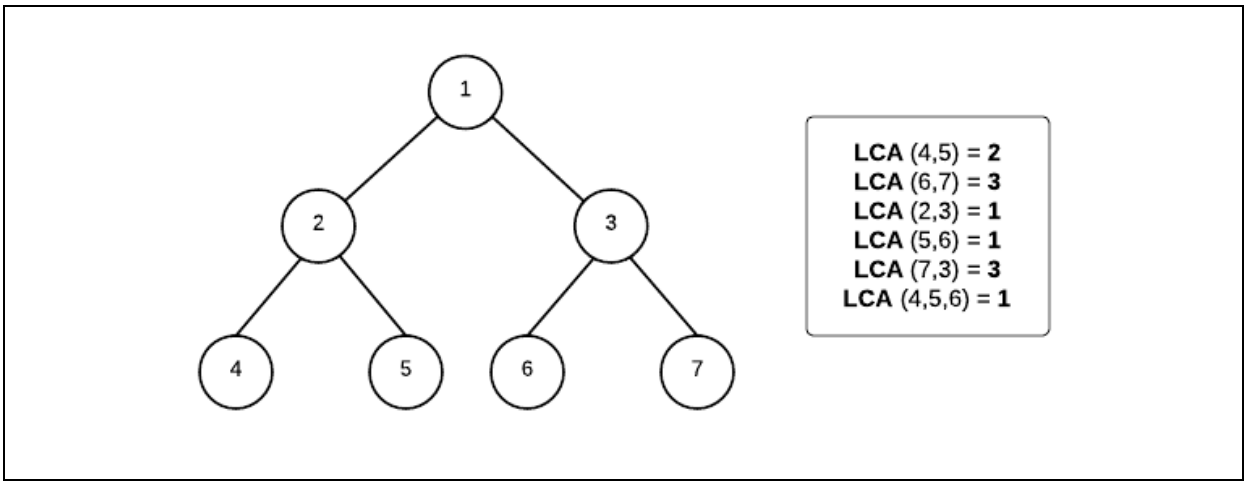


Figura 13 - Exemplo de LCA

Para que os pesquisadores possam utilizar o LCA em suas pesquisas, o programa foi também disponibilizado para acesso via linha de comando e por meio de *Web services*. Para acesso via linha de comando, foram criados dois *scripts*. O primeiro, denominado *LCARunner*, lê um arquivo contendo uma lista de *tax ids* (um em cada linha) e escreve na saída padrão o *tax_id* e nome científico do LCA calculado. A Figura 14 demonstra um exemplo do *log* de saída do *script LCARunner*. Nela é possível visualizar o passo-a-passo de execução do programa, desde o carregamento da árvore até o resultado final, que denota o LCA encontrado (*Mammalia* no exemplo).

```
05/04/2014 03:54:00 DEBUG [main] Building full taxonomy tree. (TaxonomyTree.groovy:26)
05/04/2014 03:54:00 DEBUG [main] Retrieving all database taxon entries. (TaxonomyTree.groovy:28)
05/04/2014 03:54:16 DEBUG [main] Retrieving merged entries... (TaxonomyTree.groovy:33)
05/04/2014 03:54:16 DEBUG [main] 1127870 taxons have been retrieved. (TaxonomyTree.groovy:37)
05/04/2014 03:54:16 DEBUG [main] Creating root node. (TaxonomyTree.groovy:42)
05/04/2014 03:54:16 DEBUG [main] Root node successfully created. (TaxonomyTree.groovy:51)
05/04/2014 03:54:16 DEBUG [main] Filling tree. (TaxonomyTree.groovy:52)
05/04/2014 03:54:20 DEBUG [main] Filling depths... (TaxonomyTree.groovy:116)
05/04/2014 03:54:21 DEBUG [main] Tree successfully built. (TaxonomyTree.groovy:123)
05/04/2014 03:54:21 DEBUG [main] Calculating the LCA for taxId list. (LCA.groovy:81)
05/04/2014 03:54:21 DEBUG [main] LCA Found. Tax id: 40674 (Mammalia). (LCA.groovy:110)
```

Figura 14 - Log do programa de linha de comando *LCARunner*

O *script LCARunner* é útil para o cálculo do LCA de um grupo de *taxa* isolado. No entanto, algumas vezes o pesquisador pode necessitar descobrir os ancestrais comuns de vários grupos de *taxa* ao mesmo tempo. Para esses casos, foi desenvolvido o segundo *script* de linha de comando*,* denominado *MultiLCA*. Esse *script* processa um lote de arquivos presentes em um mesmo diretório, cada qual contendo uma lista de *tax ids*. O programa retorna um novo arquivo com os resultados, sendo um LCA por linha.

Para os casos em que o pesquisador necessita integrar o módulo de LCA em um programa existente de forma remota, foi produzido um *Web service,* disponível tanto em formato SOAP quanto REST [25].

Para acesso SOAP, o usuário deve utilizar o WSDL disponível em http://biodados.icb.ufmg.br:8080/BioToolsService/services/lca?wsdl. O *Web service* oferece um método denominado *lca* que recebe como parâmetro uma lista de *tax ids*. O retorno é um objeto que descreve o táxon que é o LCA dos *tax ids* passados por parâmetro.

Para acesso REST, o endereço a seguir deve ser acessado a partir de um *Web* browser:

```
http://merengue.icb.ufmg.br:8080/BioToolsService/lca/txid1+txid2+txid3+…
```

no qual *txidX* são os *tax ids* do conjuntos de *taxa* que se deseja calcular o LCA. Os *tax ids* devem ser separados pelo caractere '+', como no exemplo a seguir:

```
http://merengue.icb.ufmg.br:8080/BioToolsService/lca/9606+9783
```

A chamada acima retorna o LCA calculado na árvore taxonômica entre os organismos *Homo sapiens (*homem, *tax id* 9606) e *Elephas maximus* (elefante asiático, *tax id* 9783).Para os *Web services* do tipo REST, o retorno é um arquivo no formato XML com *tags* que descrevem o LCA dos *tax ids* passados por parâmetro. A Figura 15 exibe o conteúdo do arquivo no formato XML retornado para o exemplo acima. Além das *tags* que descrevem o táxon LCA, pode-se notar também a *tag firstRanked.* Essa *tag* indica o mais baixo nodo na linhagem do LCA que é categorizado com um *rank* válido. No exemplo, o LCA resultante é o clado *Eutheria*, o qual é categorizado como *"no rank".* O nodo ranqueado mais próximo de *Eutheria* é *Mammalia,* categorizado com o *rank* classe. Portanto, esse nodo também é retornado na resposta do *Web service.*

Como pode ser observado, o uso de *Web services* do tipo REST tem a desvantagem de que o usuário deve fazer o *parsing* do arquivo XML retornado em seu código. Porém, para consultas rápidas via *browser,* este é o método mais indicado.

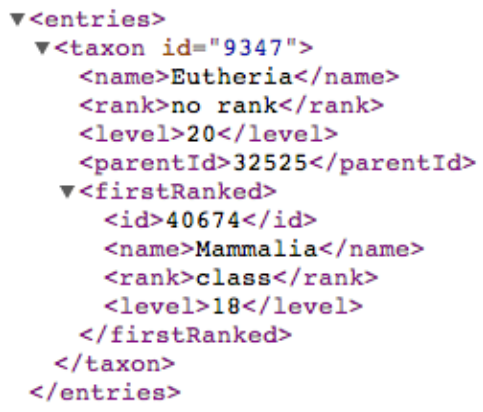


Figura 15 - Exemplo de arquivo XML retornado pelo Web service REST

Os serviços apresentados acima e a geração da tabela *TaxSimple* atualizada fazem parte do um pacote básico de ferramentas bioinformáticas ofertadas neste trabalho. Nas próximas seções são apresentados produtos desenvolvidos neste projeto que fazem uso dessas ferramentas básicas para a geração de resultados de relevância biológica

4.5 *GeneOrigin*

Um grupo de ortólogos é composto por variantes de um mesmo gene que possuem origem em um ancestral comum. A identificação do ancestral que deu origem a um grupo de ortólogos fornece uma boa indicação do clado taxonômico de origem do gene correspondente. A KEGG (*Kyoto Encyclopedia of Genes and Genomes*) [15] disponibiliza uma base atualizada de grupos de ortólogos, curados manualmente a partir de genomas completos. Essa base, denominada KO (KEGG *Orthology)*, é composta de grupos de ortólogos KO identificados por uma sequencia K#####, na qual # representa um número. Cada grupo de ortólogos KO representa, portanto, um gene.

Como explicado anteriormente, a base de dados KO foi construída exclusivamente a partir de organismos que já possuem o genoma completamente sequenciado. Sabe-se que o sequenciamento é um processo lento com as técnicas atuais e o número de espécies na árvore da vida é absurdamente grande. Portanto, mesmo com as frequentes atualizações, o crescimento da base de dados da KEGG é lento, pois depende da velocidade em que genomas

completos se tornam disponíveis e que são incluídos na referida base. Dessa forma, neste trabalho foi utilizada como base de grupos de ortólogos uma versão enriquecida da base de dados KO, denominada UEKO. Na versão utilizada neste trabalho, o enriquecimento causa a elevação no número de ortólogos de 1.411.402 na base original KO para 2.881.880 em UEKO, um aumento de aproximadamente 104%. A construção da base de dados UEKO é explicada com maiores detalhes na seção 2.6. É importante notar que o enriquecimento não altera a designação do grupo, que permanece com o mesmo identificador da base KO.

A metodologia usada para a inferência da origem de um gene neste trabalho se resume a extrair dos grupos de ortólogos os organismos que tem ali seus genes representados e então aplicar sobre o conjunto de *tax ids* obtidos o algoritmo de LCA.

Para o cálculo em massa do LCA de todos os grupos KO (ou UEKO, pois são enriquecidos mais possuem o mesmo identificador) presentes na base de dados, foi criado um módulo denominado *UekoLCA*, que pode ser importado para dentro de outros projetos. Basicamente o módulo possui apenas uma operação *findLCAforKO,* a qual recebe como parâmetro um identificador de KO (no formato K#####) e retorna um objeto com as informações do táxon correspondente ao LCA encontrado. Os materiais e métodos utilizados para implementação do módulo *UekoLCA* são descritos na seção 3.6.

Para permitir acesso rápido e fácil para pesquisadores interessados na origem comum de grupos de ortólogos, foram também desenvolvidos dois *scripts* executáveis em linha de comando: *UekoLCARunner* e *UekoLCAFileRunner*.

O *script UekoLCAFileRunner* tem a finalidade de executar consultas pontuais a LCAs por meio de linha de comando. O programa encontra os LCAs de uma lista de identificadores de grupos de ortólogos KO, passados como parâmetro por meio de um arquivo de entrada. O arquivo de entrada deve conter identificadores de grupos de ortólogos KO em cada uma de suas linhas, no formato K#####. O retorno do *script* é um arquivo no formato TSV (separado por tabulação, no qual cada linha do arquivo contém um identificador KO e informações do LCA calculado para o grupo de ortólogos correspondente. A lista abaixo exemplifica a saída do *script UekoLCAFileRunner.*

```
K10590     2759       Eukaryota       2     superkingdom
K02649     33213      Bilateria       6     no rank
K10568     9347       Eutheria        20    no rank
```

No exemplo acima, a primeira coluna denota o identificador do grupo KO. As demais colunas são referentes ao LCA encontrado: a segunda representa o *tax id*, a terceira o nome científico, a quarta o nível (distância até a raiz) e a quinta o *rank.*

O segundo *script* desenvolvido para o módulo *UekoLCA,* denominado *UekoLCARunner,* tem sua abordagem a nível de uma espécie como um todo, ao invés de genes individuais. O programa encontra os LCAs de todos os grupos de ortólogos UEKO (ou KO) de um determinado organismo e grava os resultados em uma nova tabela no banco de dados. O *script* recebe como parâmetro o *tax_id* da espécie a calcular, a base de dados a consultar (KO ou UEKO) e o nome da tabela de destino. Os resultados são gravados na tabela de destino, que possui a seguinte estrutura de colunas:

- Coluna *ko* - número do grupo KO (formato K#####).
- Coluna *lca_txid* - *Tax id* do LCA encontrado para o grupo KO.
- Coluna *lca_name* - Nome científico do LCA calculado.
- Coluna *lca_level* - A distância do LCA até a raiz da árvore (em número de nodos).
- Coluna *lca_rank* - O *rank* do LCA (espécie, gênero, classe, etc.).

Para a tabela de resultados, convencionou-se a nomenclatura *xxx_base_lca*, na qual *xxx* representa o identificador de três letras da espécie e *base* é o banco de dados utilizado no cálculo, o qual pode ser *ko* ou *ueko*. A Figura 16 ilustra algumas linhas da tabela *hsa_ueko_lca*, gerada pelo *script UekoLCARunner*. Essa tabela contém os ancestrais comuns de todos os grupos de ortólogos KO da espécie humana (enriquecidos pelo algoritmo UEKO). Pelo fato de *Homo sapiens* possuir o genoma completo sequenciado, a maioria de seus genes conhecidos possuem grupos de ortólogos KO correspondentes.

ko	lca_txid	lca_name	lca_level	lca_rank
K00490	2759	Eukaryota	2	superkingdom
K01587	33208	Metazoa	4	kingdom
K12011	314146	Euarchontoglires	21	superorder
K12013	7711	Chordata	9	phylum
K12015	314146	Euarchontoglires	21	superorder
K12018	314146	Euarchontoglires	21	superorder
K04237	33208	Metazoa	4	kingdom
K12023	314146	Euarchontoglires	21	superorder
K12029	314146	Euarchontoglires	21	superorder
K12027	314146	Euarchontoglires	21	superorder

Figura 16 - Linhas da tabela *hsa_ueko_lca*

4.6 Padrões de surgimento de genes em uma espécie

As tabelas contendo as origens gênicas de cada espécie geradas pelo *script UekoLCARunner* abrem caminho para o entendimento dos padrão de surgimento de genes ao longo da evolução de um dado organismo. Além disso, possibilitam uma análise comparativa entre os padrões de surgimento de genes entre espécies diferentes.

Uma consulta simples da base gerada na tabela *hsa_ueko*_lca, por exemplo, possibilita o agrupamento dos dados das tabelas por clado e a subsequente contagem do número de genes em cada clado. Isso fornece uma estimativa do número de genes emergentes em cada um deles.

A Figura 17 mostra os dados da *view hsa_origin_ueko,* gerada a partir do agrupamento descrito acima. Observação: genes viróticos não foram incluídos, a fim de se evitar desvios causados por transferência horizontal – oncogenes por exemplo - e clonagens em vetores virais.

level	taxa	gene_count	level	taxa	gene_count
1	cellular organi...	744	16	Tetrapoda	349
2	Eukaryota	1234	17	Amniota	152
3	Fungi/Metazo...	173	18	Mammalia	3
4	Metazoa	182	19	Theria	9
5	Eumetazoa	309	20	Eutheria	453
6	Bilateria	167	21	Euarchontoglires	408
7	Coelomata	260	22	Primates	0
8	Deuterostomia	34	23	Haplorrhini	0
9	Chordata	103	24	Simiiformes	0
10	Craniata	3	25	Catarrhini	5
11	Vertebrata	15	26	Hominoidea	0
12	Gnathostomata	47	27	Hominidae	2
13	Teleostomi	0	28	Homininae	2
14	Euteleostomi	1119	29	Homo	0
15	Sarcopterygii	3	30	Homo sapiens	35

Figura 17 - *View hsa_origin_ueko (dividida em 2 partes)*

Pelo mapeamento desses dados em gráficos, é possível se ter uma boa perspectiva da maneira como os genes humanos surgiram ao longo da sequencia de clados. A Figura 18 mostra os gráficos gerados com base nos dados da *view hsa_origin_ueko*. No eixo X está mapeada a linhagem de clados de *Homo sapiens*, da raiz à folha da árvore taxonômica. O eixo Y exibe a quantidade de genes originados em cada clado. É importante notar que quando um gene possui múltiplos parálogos, ele é contado apenas uma vez, pois a análise é baseada em

grupos KO. Foram analisados 6307 grupos KO, que enriquecidos (UEKO), englobam um total de 25139 proteínas.

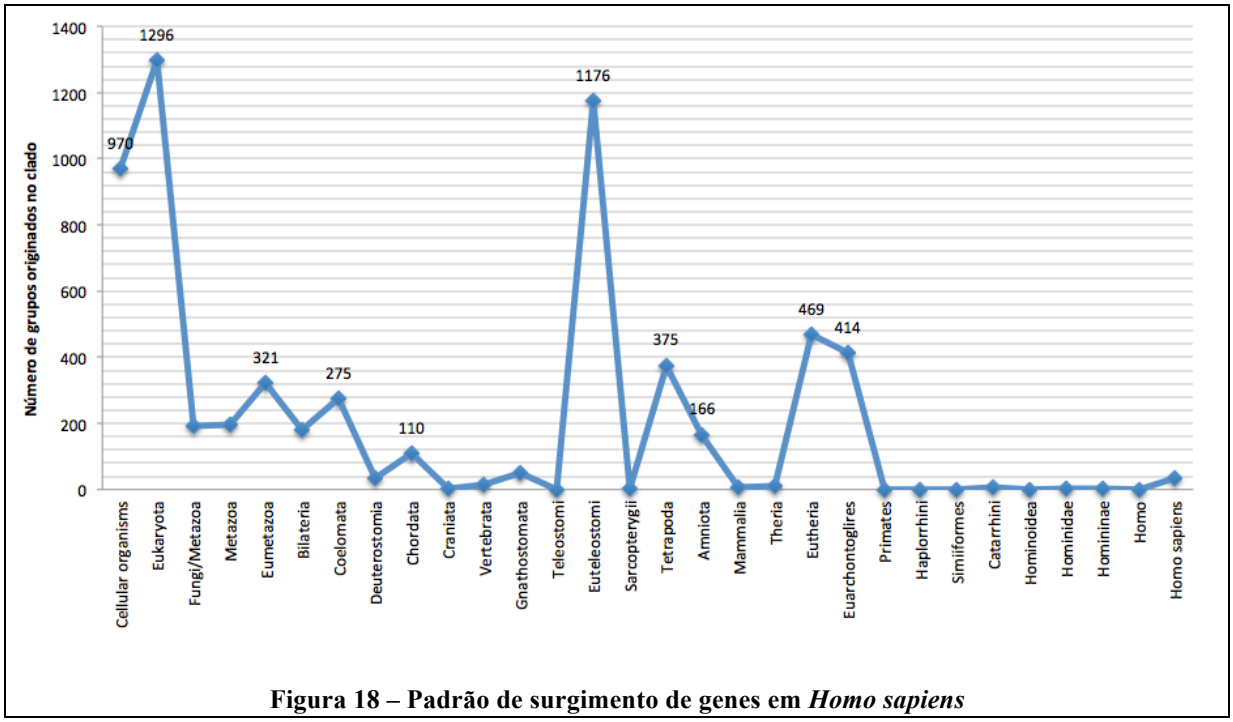


Figura 18 – Padrão de surgimento de genes em *Homo sapiens*

Pela análise da Figura 18, é possível notar que em alguns clados houve uma explosão de surgimento de novos genes, criando no gráfico um padrão ondular. Nota-se que particularmente em *Eukaryota* (20% do total de genes), *Euteleostomi* (17%), *Tetrapoda/Amniota* (11%) e *Eutheria/Euarchontoglires* (19%) ocorre um surgimento extremamente acentuado de genes em comparação com os demais clados. Pelo que se sabe, esta é a primeira vez que se tenta retratar essa evolução.

4.7 LCA de funções moleculares e processos biológicos

A análise dos gráficos de LCA mostra picos de surgimento de genes durante a evolução das espécies. Como o surgimento de genes tende a refletir no aparecimento de novas funções moleculares e processos biológicos, é de interesse também a realização de uma análise similar em um banco de dados contendo esse tipo de informação. O objetivo é descobrir se houve também picos de surgimento de funções e processos, ou se isso ocorreu apenas com genes, além de se reproduzir o fenômeno com uma base diferente. Para isso, foi realizado um processo similar ao executado para UEKO na base de dados UniProtKB-GOA (*Gene Onthology Annotation*).

O projeto *Gene Onthology* (GO) [40] é um esforço colaborativo que objetiva a criação de descrições não-redundantes para os produtos de genes. A base de dados GO é estruturada na forma de árvore hierárquica e é composta basicamente por termos, os quais descrevem produtos de genes com relação a seus processos biológicos, componentes celulares associados e funções moleculares. Os termos não dão nomes a objetos biológicos como genes e proteínas, mas sim, descrevem fenômenos moleculares associados a eles (por exemplo, "apoptose"). Esse tipo de estrutura é denominado uma ontologia, cujo objetivo é prover um conjunto conhecido de termos para descrever o conhecimento a respeito de um assunto. O *GO Consortium* (consórcio que administra a base de dados GO) realiza as anotações de termos GO, isso é, associa produtos de genes a termos GO que os descrevem.

O projeto UniProtKB-GOA [41] faz parte também do *GO Consortium* e seu objetivo é associar anotações GO a proteínas presentes no banco de dados UniProt [19]. No momento da escrita deste manuscrito, o projeto anunciava que disponibilizava mais de 100 milhões de anotações englobando 11 milhões de proteínas. A base é atualizada a cada quatro semanas.

O UniProtKB-GOA disponibiliza para *download* um arquivo no formato TSV (valores separados por tabulações) contendo a base completa de mapeamentos entre termos GO e proteínas. O arquivo também possui uma coluna que informa se o termo está associado a um processo biológico, função molecular ou componente celular. Além disso, outra coluna identifica a espécie portadora da proteína, por meio de seu *tax id*. O arquivo foi carregado em um banco de dados local, a fim de possibilitar consultas na linguagem SQL.

Para realizar a análise da origem de processos biológicos e funções moleculares, foi criado um módulo denominado GoaLCA. Esse programa recebe como parâmetro um filtro de domínio, que identifica se a busca será feita em processos ou funções (componentes celulares não foram analisados), e um filtro de *tax id*. O programa então calcula o LCA de cada termo GO associado ao domínio e ao *tax id* passados por parâmetro. O conjunto de entrada para cálculo do LCA são os *tax ids* de todas as espécies que possuem funções ou processos associados a um determinado termo GO.

Os resultados são gravados em uma tabela que se convencionou chamar de *xxx_goa_D_lca*, no qual *xxx* é a sigla de três letras da espécie em análise e D pode assumir o valor *f* (para funções moleculares) ou *p* (para processos biológicos), dependendo do domínio de termos GO pesquisado. Para *Homo sapiens* foram geradas, portanto, as tabelas *hsa_goa_f_lca* e *hsa_goa_p_lca*. De modo similar ao executado para a análise da ancestralidade de grupos de ortólogos, os dados de cada tabela foram agrupados por clado, e a quantidade de funções ou processos somados para cada um deles. As tabelas

hsa_origin_goa_f e *hsa_origin_goa_p* foram geradas a partir desse agrupamento. Os dois gráficos gerados a partir dessas tabelas foram sobrepostos ao gráfico gerado por meio da tabela UEKO e exibidos na Figura 19. Foram analisados 6307 grupos KO enriquecidos (UEKO), 2996 termos GO para função e 5574 termos GO para processo.

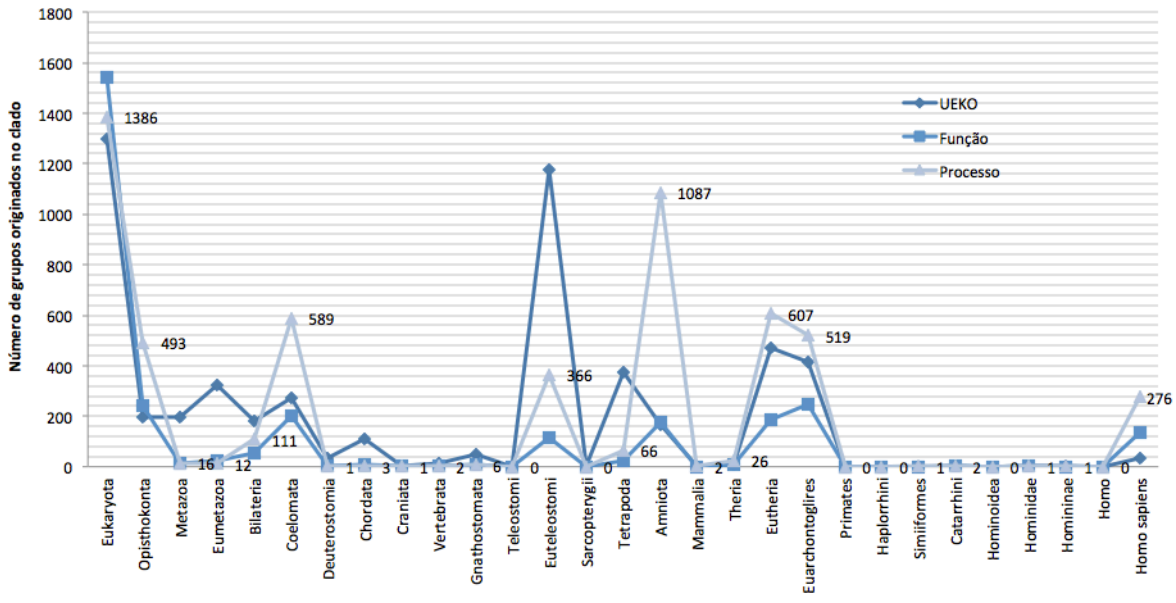


Figura 19 - *Homo sapiens:* Surgimento de gripos de ortólogos x funções moleculares x processos biológicos

Pode-se observar que os padrões de onda encontrados para o surgimento de genes também são visíveis para processos e funções, e os picos aparecem aproximadamente nos mesmos clados. Uma diferença visível aparece com relação ao tamanho dos picos. Em UEKO, o maior pico ocorre em *Euteleostomi* enquanto em processos GO ocorre em *Amniota*. Já em *Eutheria* /*Euarchontoglires* esses dois gráficos tem curvas bem parecidas.

Pela análise dos gráficos pode-se concluir que, apesar de não ser evidente uma relação direta nas quantidades de genes, funções moleculares e processos biológicos emergentes, nota-se que certos clados apresentam papel diferenciado no surgimento dos mesmos.

4.8 LCA dos genes, funções e processos em outras espécies

A fim de se esclarecer se o surgimento dos genes em ondas é uma característica exclusiva da linhagem humana, foram gerados gráficos similares para outras quatro espécies de ramos bem distantes na árvore taxonômica: a mosca *Drosophila melanogaster*, a planta herbácea *Arabidopsis thaliana*, a gramínea *Oryza sativa* (arroz) e a bactéria *Escherichia coli*.

A Figura 20 apresenta o gráfico gerado para *Drosophila melanogaster*. Foram analisados 2700 grupos KO enriquecidos (UEKO), 1617 termos GO para função e 2545 termos GO para processo. Pode-se observar que o pico em *Coelomata* se repete. No entanto, poucos genes, funções e processos aparecem a partir daí, com exceção de um pico de processos que surge apenas no clado folha. Tal apresentação sugere que o número de grupos de ortólogos contendo genes da moscas nas bases analisadas pode não ser suficiente. Todavia, aqueles presentes parecem ser mais ancestrais.

A Figura 21 mostra que o surgimento de novos processos e funções em *Drosophila melanogaster* segue padrão semelhante ao do surgimento de genes.

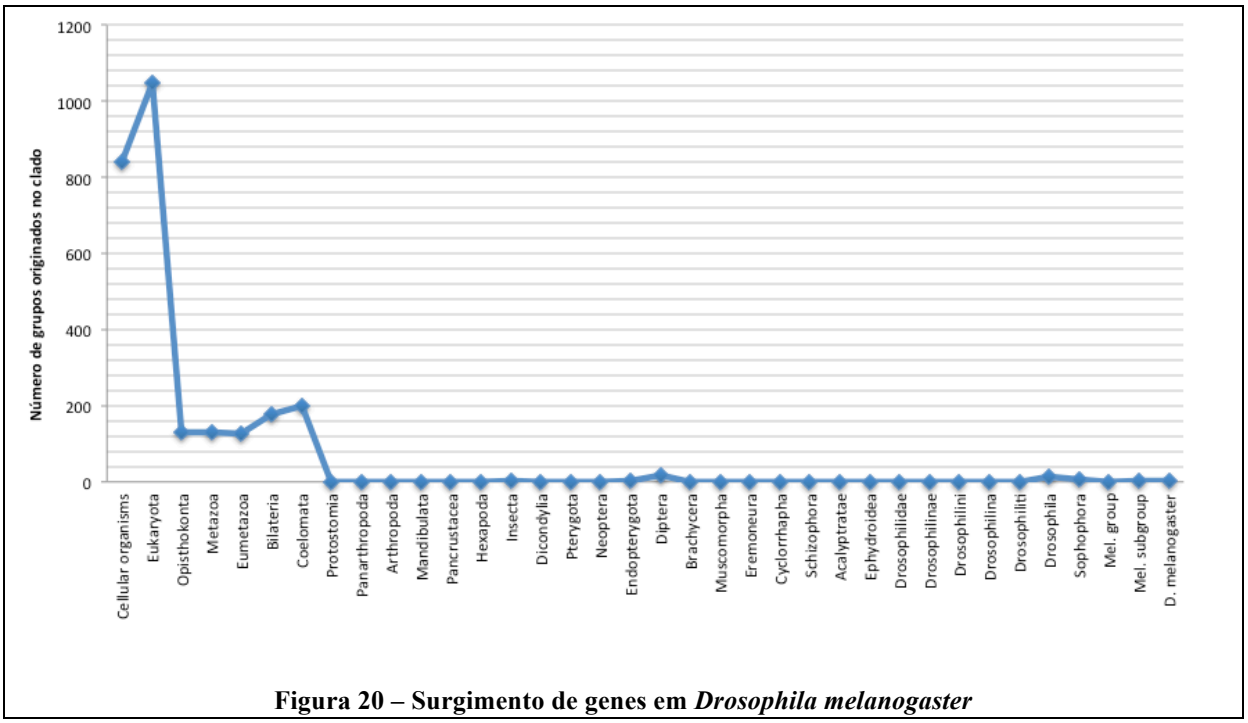


Figura 20 – Surgimento de genes em *Drosophila melanogaster*

A Figura 22 exibe o gráfico gerado para *Arabidopsis thaliana.* A análise foi feita apenas nas bases de dados UEKO. Foram analisados 2182 grupos KO enriquecidos (UEKO).

É possível observar que, assim como nas espécies animais analisadas anteriormente, existem também clados, como *Magnoliophyta,* que apresentam surgimento superior de genes. Entretanto, assim como em *Drosophila melanogaster,* o gráfico mostra a maioria absoluta dos mesmos surgindo em clados iniciais da evolução.

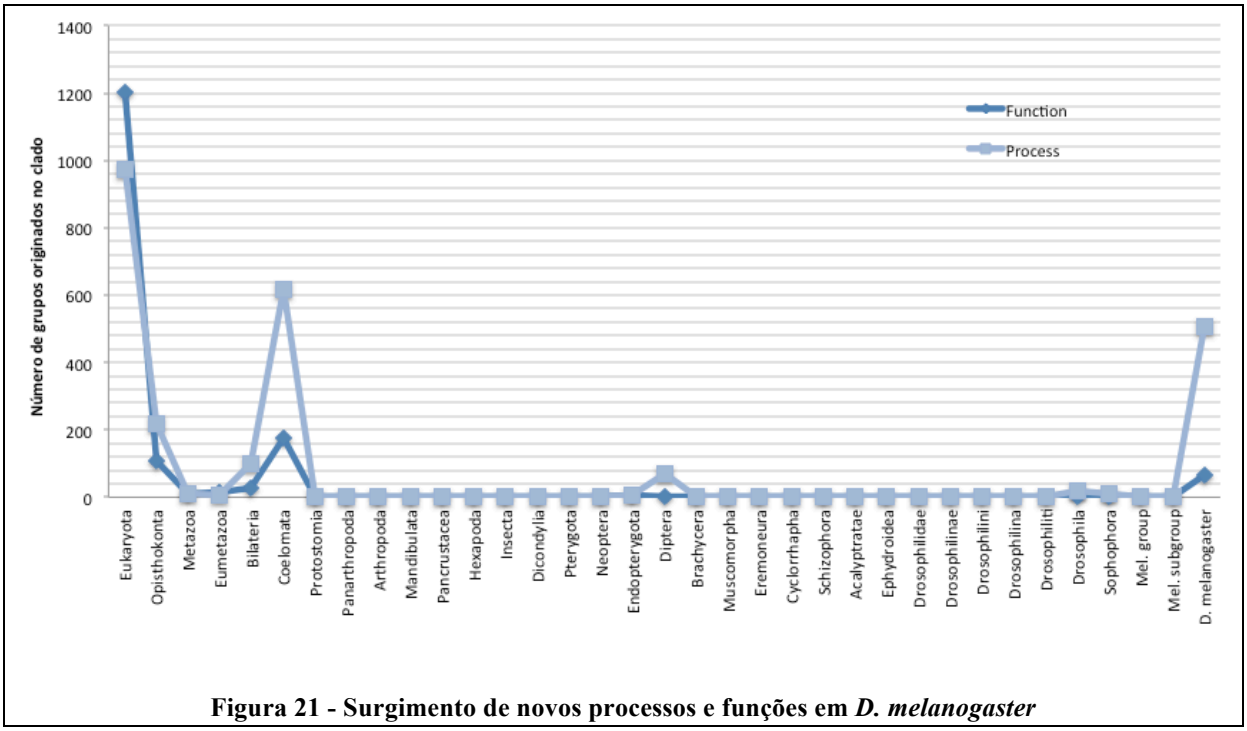


Figura 21 - Surgimento de novos processos e funções em *D. melanogaster*

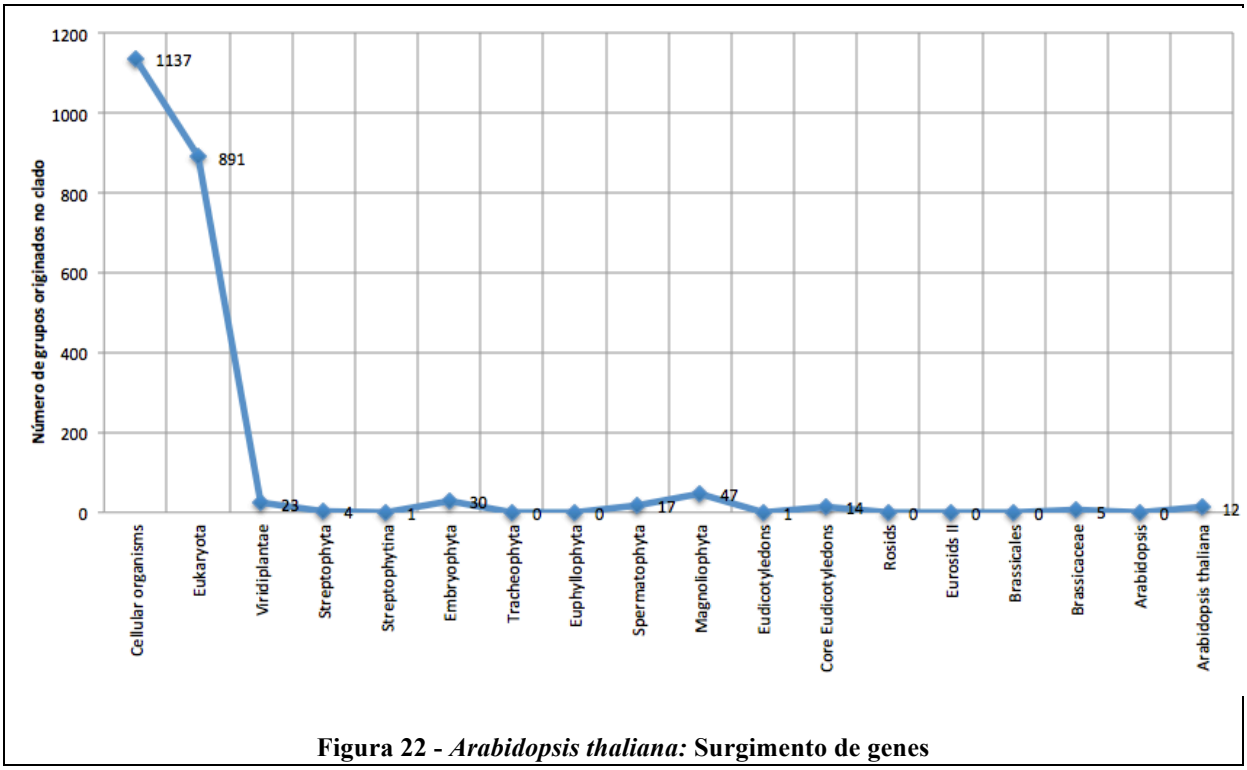


Figura 22 - *Arabidopsis thaliana:* Surgimento de genes

A segunda planta analisada é a gramínea *Oryza sativa,* popularmente conhecida como arroz. A Figura 23 apresenta o gráfico gerado, que é similar ao construído para *Arabidopsis thaliana.*

A Figura 24 apresenta o gráfico de surgimento de genes gerado para *Escherichia coli.* Nesse caso, apenas a base UEKO foi utilizada. Foram analisados 2584 grupos KO enriquecidos (UEKO). Nota-se que a bactéria também teve a maioria absoluta de seus genes surgindo nos primeiros clados. Um pico menor ocorre também em *Enterobacteriaceae.*

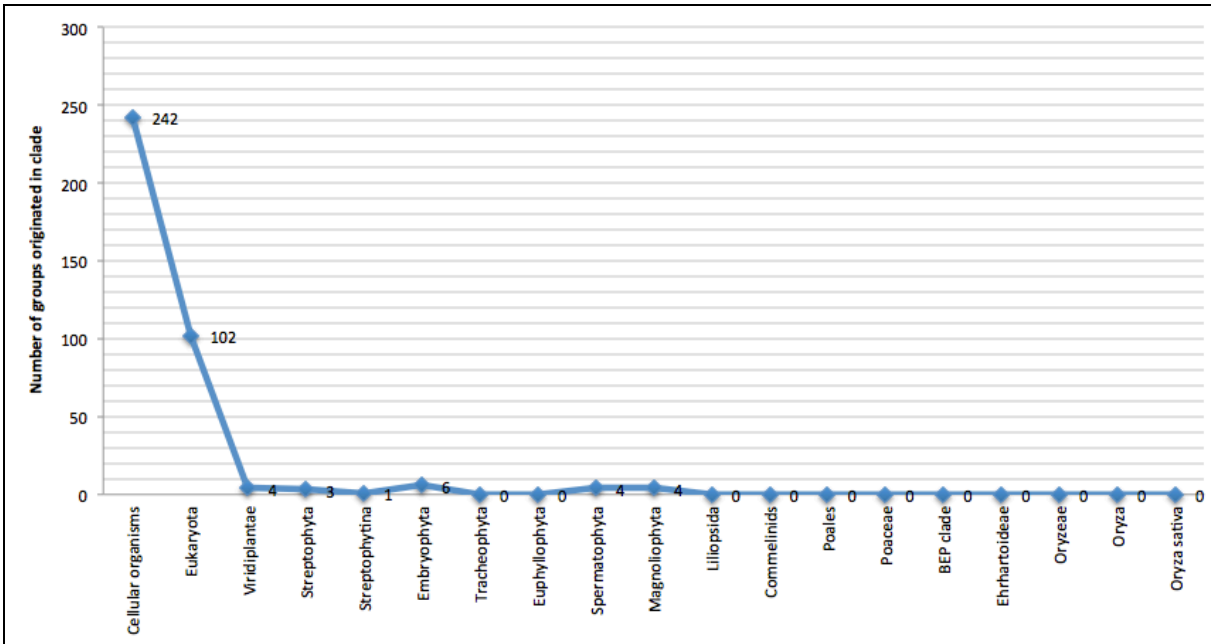


Figura 23 - *Oryza sativa:* surgimento de genes

A análise dos gráficos construídos mostra que o padrão ondular de surgimento de genes, funções moleculares e processos biológicos, se repete para as diferentes espécies. O *website* Genesis, criado recentemente e apresentado no capítulo 5 deste trabalho, permite a inspeção da origem de grupos KO de todas as espécies. Estudos adicionais deverão explorar em maior detalhe os genes e sistemas que estão surgindo nas épocas indicados por Genesis.

4.9 Origem das vias metabólicas humanas

O estudo da origem dos genes também permite a análise da ancestralidade dos sistemas dos quais eles participam. Os genes dão origem às enzimas que são as peças fundamentais das vias metabólicas dos organismos. Por isso, o conhecimento das origens gênicas fornece base para o estudo do surgimento e da evolução das vias metabólicas.

Vias metabólicas são uma série de reações químicas que ocorrem dentro de uma célula. Os produtos de uma reação servem como substrato para as seguintes e assim por diante. Geralmente, a função principal de uma via é a transformação de compostos químicos,

a fim de gerar novas biomoléculas necessárias para o funcionamento da célula. Existem, por exemplo, vias catabólicas e anabólicas, que objetivam quebrar ou sintetizar novas moléculas, respectivamente. Vias podem ser integradas umas nas outras, de forma a gerar complexas redes metabólicas que atuam no funcionamento da célula e do organismo de uma forma geral. Um exemplo dessa complexidade são as vias que fazem a manutenção da homeostase do organismo.

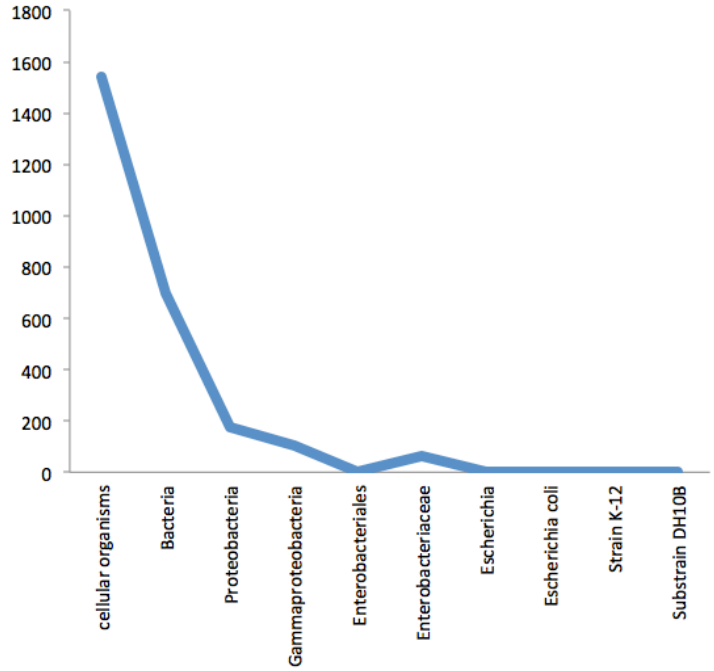


Figura 24 - *Escherichia coli:* Surgimento dos genes

Neste trabalho, a base de dados KEGG PATHWAY [15] foi utilizada juntamente com as informações relativas à origem dos genes para a coleta de dados evolutivos importantes sobre as vias metabólicas humanas. A via de sinalização JAK-STAT foi escolhida para uma análise aprofundada. Um passo-a-passo explica como as proteínas foram interagindo ao decorrer da evolução até a formação da via como é conhecida atualmente.

A KEGG disponibiliza em sua base de mapas de vias metabólicas arquivos do tipo KGML (*KEGG Markup Format*), os quais conformam com o formato XML e possuem um conjunto de *tags* especializado na descrição de vias metabólicas. Há *tags* que identificam os principais elementos que compõem uma via: enzimas e demais compostos químicos envolvidos. Também há *tags* que identificam relações entre enzimas e *tags* que identificam

reações químicas entre compostos. O arquivo KGML é utilizado na construção automática dos mapas de vias metabólicas disponíveis no site da KEGG.

Para o objetivo deste trabalho, os arquivos KGML de todas as vias metabólicas humanas foram transferidos para um servidor local. Foi criado um *script* denominado KGMLtoDB para fazer o *parsing* desses arquivos de modo a carregar as informações relevantes para o banco de dados local. O *script* gera duas tabelas. A tabela *path_entry* armazena os elementos que compõe a via e seus respectivos tipos, que podem ser "gene", "composto" e outros menos utilizados. A tabela *entry_relationship* armazena as relações entre os elementos, par a par. As relações podem ser do tipo enzima-composto, enzima-enzima e outras. O *script* ainda associa os grupos de ortólogos KO (através do número K) aos genes inseridos na tabela *path_entry*. Sabendo-se a origem dos elementos da via, pode-se então analisar a evolução da mesma, como será descrito abaixo. Mais detalhes sobre a implementação do script KGMLtoDB são descritos na seção 3.7.

4.9.1 *Análises quantitativas do número de genes das vias*

Com as informações relativas às vias e seus elementos carregadas no banco de dados local, é possível a realização de análises quantitativas gerais sobre esses dados. O histograma apresentado na Figura 25 apresenta uma visão geral do tamanho das vias humanas na base KEGG. O eixo X denota a quantidade de genes (representados na forma de grupos KO) em intervalos de 5. O eixo Y representa a quantidade de vias metabólicas. É possível observar que aproximadamente 52% das vias possuem no máximo 30 genes.

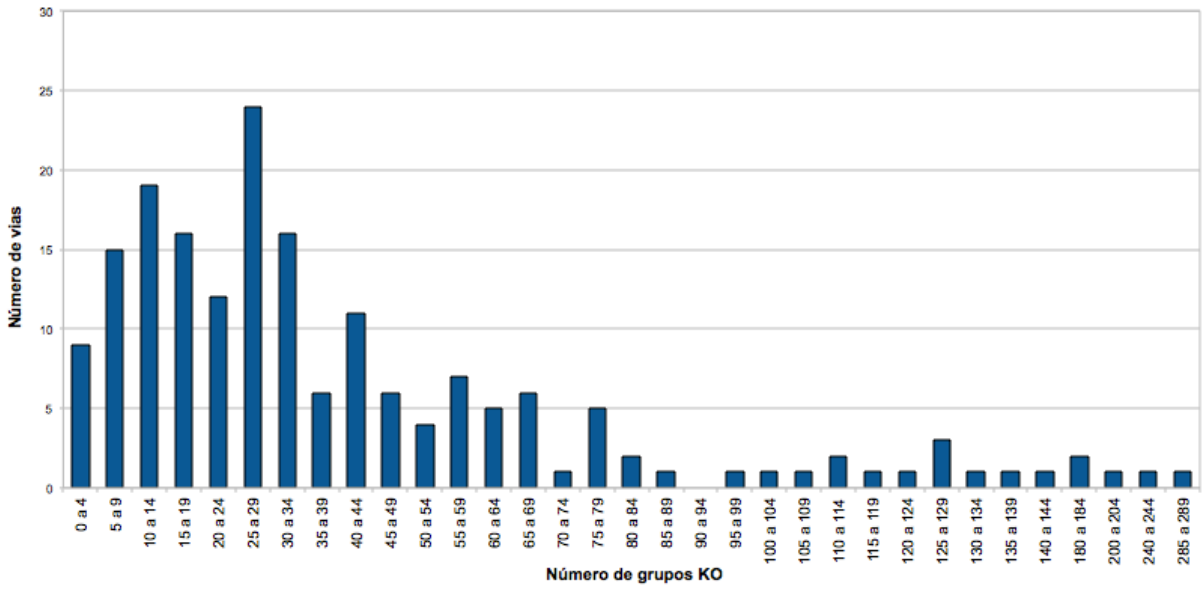


Figura 25 - *Homo sapiens:* Histograma do número de grupos KO x número de vias

Utilizando-se a tabela *tax_simple* é possível se obter, em uma simples consulta, todos os *tax ids* pertencentes a um determinado clado. Desse modo, também é viável obter o mesmo gráfico gerado anteriormente para todo o reino *Metazoa*. O gráfico é exibido na Figura 26. Os resultados são próximos aos obtidos para *Homo sapiens*. Aproximadamente 49% das vias possuem até 30 genes.

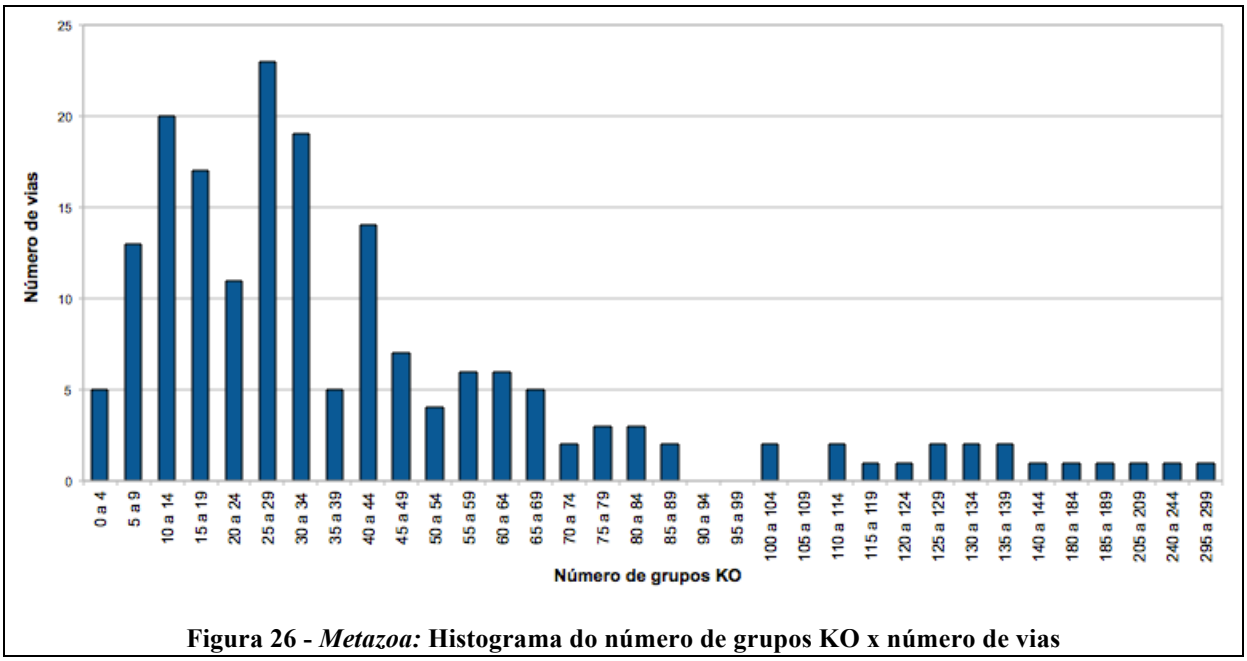


Figura 26 - *Metazoa:* Histograma do número de grupos KO x número de vias

4.9.2 *Subvias*

Outro estudo realizado teve como objetivo analisar o modo como as vias se originaram. Algumas vias podem ter surgido a partir de subvias menores compostas por uma ou duas enzimas, que durante a evolução foram agregando outras enzimas, compostos e funcionalidades, até se tornarem vias maiores. Outras vias podem ter surgido a partir da união de subvias menores. Uma subvia é definida neste trabalho como um subconjunto de enzimas de uma via metabólica que existia antes da via se tornar completa e funcional. Evidentemente, subvias poderiam ter outras funções, diferentes das que a via resultante desempenha atualmente.

Neste trabalho foi construído um programa para detecção, para cada via metabólica, de todas as subvias que existiam durante cada clado da linhagem humana. Foram construídas duas novas tabelas: *path_subgraph* e *subgraph_element*. A primeira delas tem como objetivo armazenar o número de subvias que uma determinada via possuía até um determinado clado.

Uma pergunta que pode ser respondida por essa tabela é, por exemplo, "até *Coelomata*, quantas subvias da via de metabolismo da purina já existiam, mas ainda não estavam relacionadas entre si"? A tabela *subgraph_element* armazena os elementos (enzimas e compostos) das subvias. O materiais e métodos utilizados para a construção dessas tabelas podem ser encontrados na seção 3.8.

A partir dos dados da tabela *path_subgraph*, foi possível obter informações importantes sobre a evolução das vias metabólicas humanas. Do total de 156 vias analisadas (selecionadas apenas aquelas que chegarão a ser completas em *Metazoa*, a fim de se evitar vias típicas de plantas, por exemplo), apenas 48 são formadas por uma única subvia atualmente. Isso significa que as demais 108 vias não possuem todos os seus elementos interligados. Essa pode ser uma indicação de que cada uma dessas 108 vias na realidade representa mais de uma via. É importante lembrar que a análise das vias foi baseada nos gráficos KGML. É possível que, em muitos desses 108 casos, os pesquisadores da KEGG tenham representado em um mesmo gráfico mais de uma via metabólica real. Alternativamente, faltariam elementos comunicantes, o que se pretende investigar com ferramentas de mineração de texto em outro projeto do Laboratório de Biodados.

Desse modo, por questões de simplicidade, neste trabalho optou-se por analisar a evolução das subvias nas 48 vias que são totalmente interligadas. Das 48 vias, 25 (52%) já possuíam todos os seus genes em *Vertebrata,* sete (15%) em *Eukaryota* e cinco (10%) em *cellular organisms.* Esse resultado sugere que mais da metade das vias metabólicas humanas já estavam completas antes da origem dos vertebrados. A Figura 27 apresenta um esquema que ilustra a porcentagem de vias completas ao longo da evolução humana.

Em 29 (60%) das 48 vias estudadas, a evolução ocorreu sempre em torno de um núcleo único (*single core*), ou seja, a via nunca apresentou mais de uma subvia. Nesses casos, a partir de um relacionamento inicial entre dois ou mais genes, outros genes foram se agregando durante os vários períodos da evolução humana, até que a via chegasse ao estágio que é conhecida na atualidade.

Nas 19 (40%) vias restantes, a via parece ter surgido a partir da união de subvias que anteriormente não eram relacionadas (*multiple core).* A Figura 28 exibe um gráfico sugerindo que, de forma geral, as vias *single core* adquiriram seus genes mais rapidamente que as vias *multiple core.*

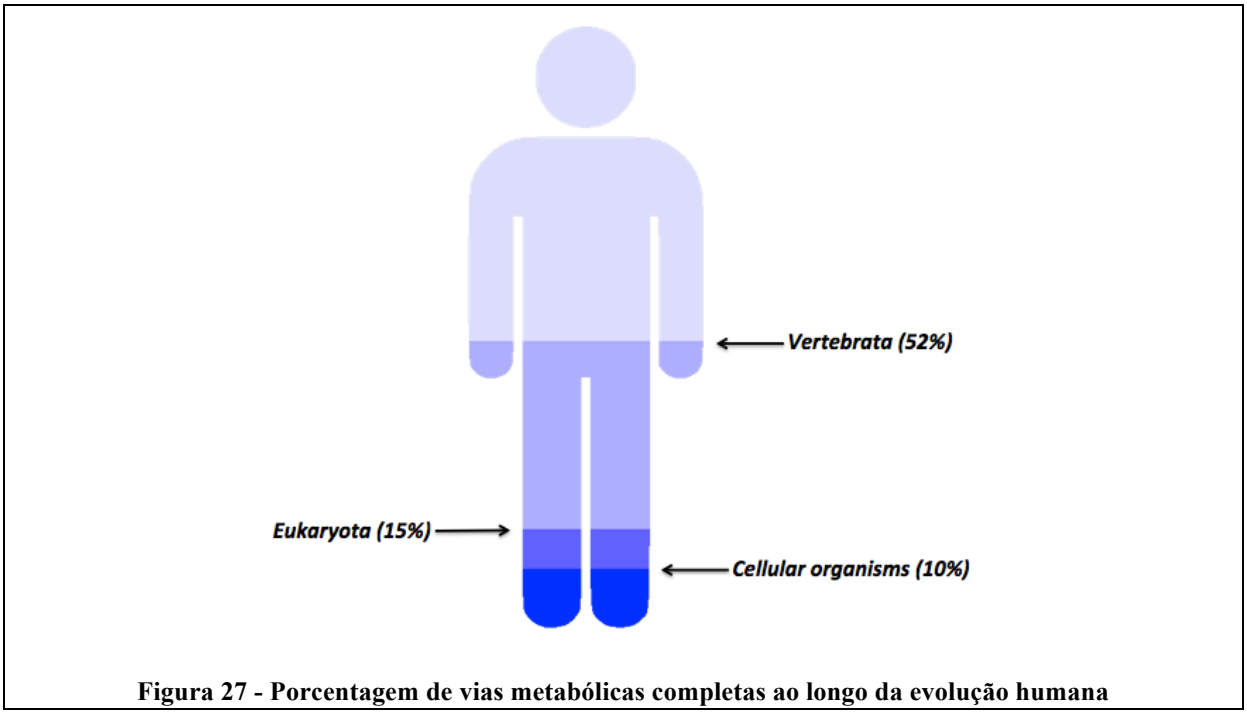


Figura 27 - Porcentagem de vias metabólicas completas ao longo da evolução humana

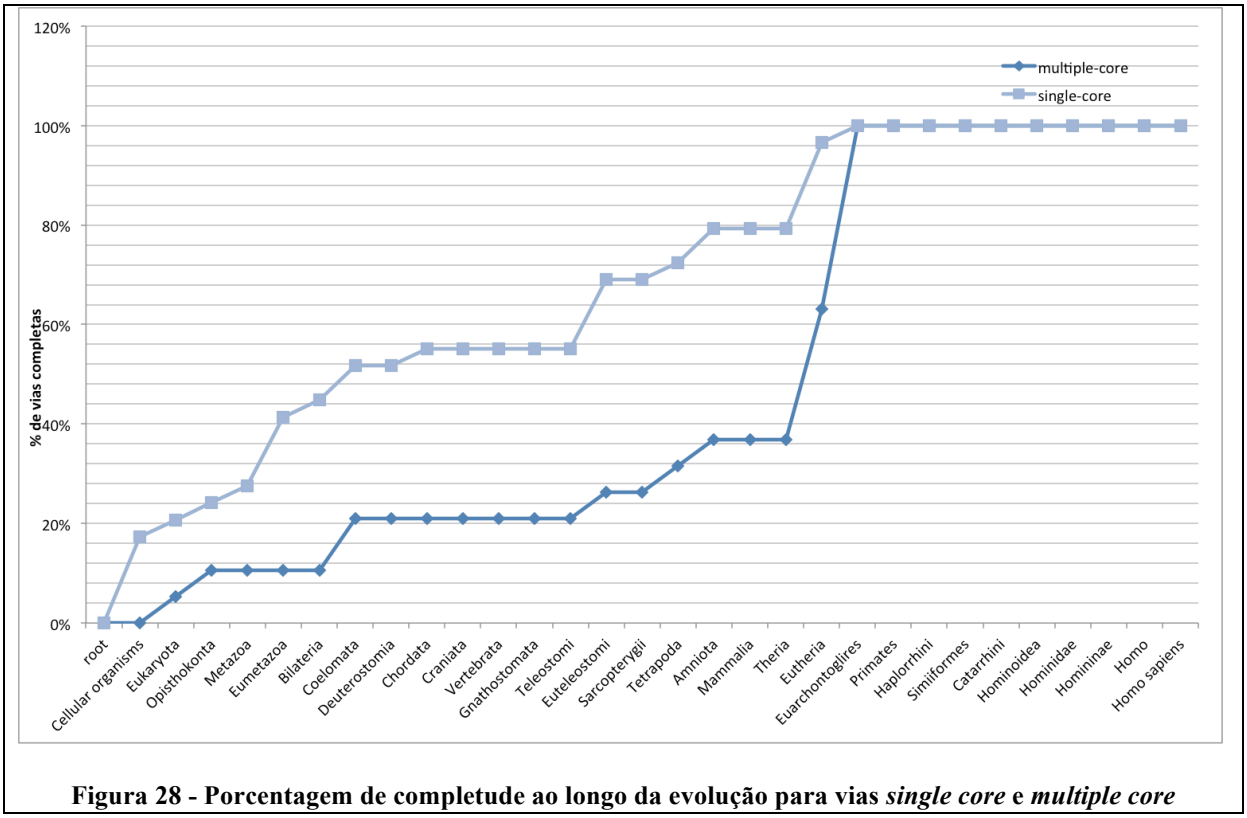


Figura 28 - Porcentagem de completude ao longo da evolução para vias *single core* e *multiple core*

Algumas vias podem aumentar ou diminuir o número de subvias ao longo da evolução. O "Sistema de Sinalização Fosfatidil Inositol", por exemplo, é originado com um único gene no clado *cellular organisms*, forma duas subvias em *Eukaryota* (sete genes), evolui para três subvias em *Opisthokonta* (10 genes), volta a ter duas subvias em *Euteleostomi*

(20 genes) e finalmente se torna uma unidade em *Eutheria* (25 genes). A via ainda ganha mais um gene antes de chegar ao estágio que é conhecida atualmente.

4.9.3 *Passo-a-passo do surgimento da via de sinalização JAK-STAT*

Por meio das informações presentes nas tabelas *path_subgraph* e *subgraph_element* é possível determinar, para uma dada via metabólica, os genes que se relacionavam em cada clado da linhagem humana. Pode-se estudar, desse modo, como as subvias foram se agregando ao longo do tempo, em uma espécie de "passo-a-passo" evolutivo.

Neste trabalho, foi escolhida a via de sinalização JAK-STAT para a realização desse estudo. A via de sinalização JAK-STAT transmite informações de sinais químicos originados fora da célula. Essa informação chega a promotores no DNA e promove a transcrição e outras atividades. Essa via evolui em torno de um núcleo único (*single core)*, ou seja, não apresenta subvias ao longo de sua evolução, sendo esse o motivo principal de sua escolha.

A Figura 29 mostra o estado da via de sinalização JAK-STAT em *Eukaryota.* Na figura, os elementos que ainda não existiam encontram-se apagados. Os genes são representados em retângulos, os processos em formas ovaladas, e as reações por setas. Desse modo, nesse estado a via possuía apenas dois genes (P13K e AKT), sendo ambos relacionados à apoptose.

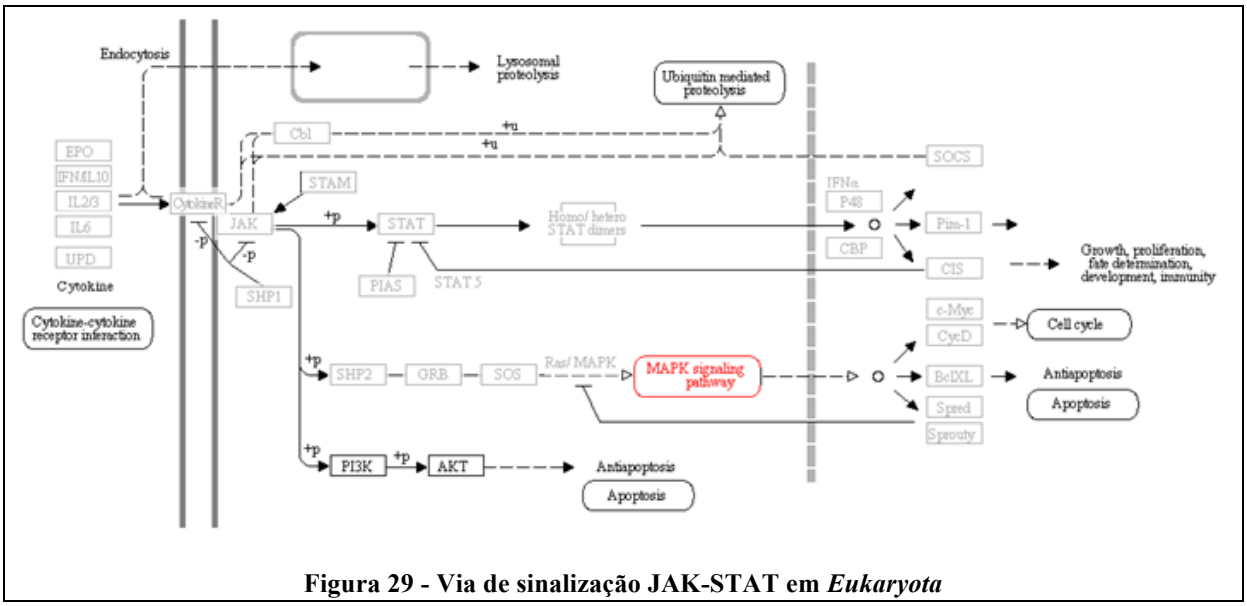


Figura 29 - Via de sinalização JAK-STAT em *Eukaryota*

Em *Metazoa*, quatro novos genes são agregados à via, incluindo JAK e STAM. A Figura 30 ilustra o estado da via em *Metazoa.*

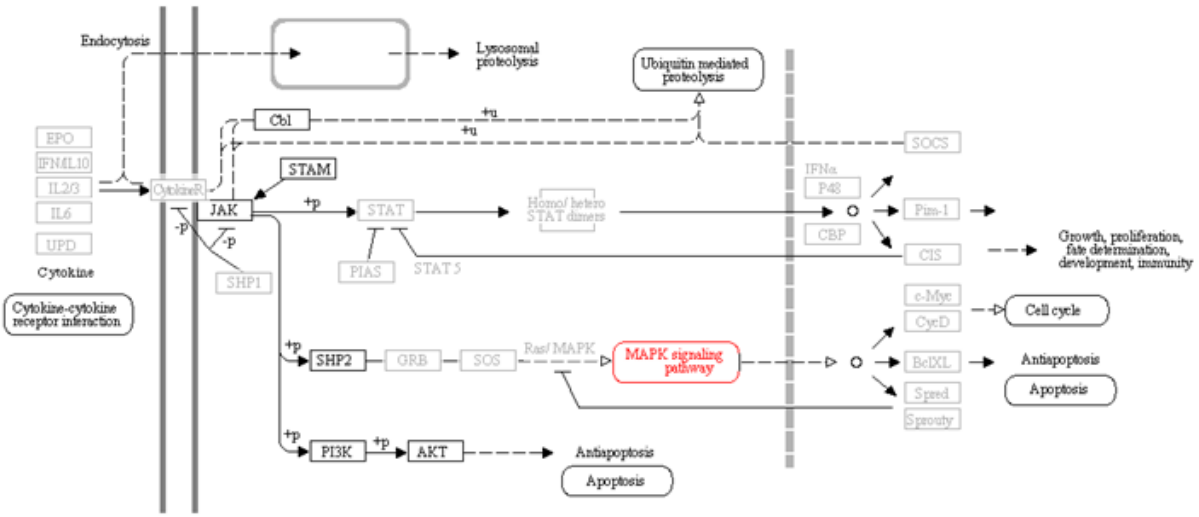


Figura 30 - Via de sinalização JAK-STAT em *Metazoa*

A Figura 31 ilustra que em *Eumetazoa,* o gene de STAT e vários outros são integrados à via, incluindo genes relacionados ao ciclo celular.

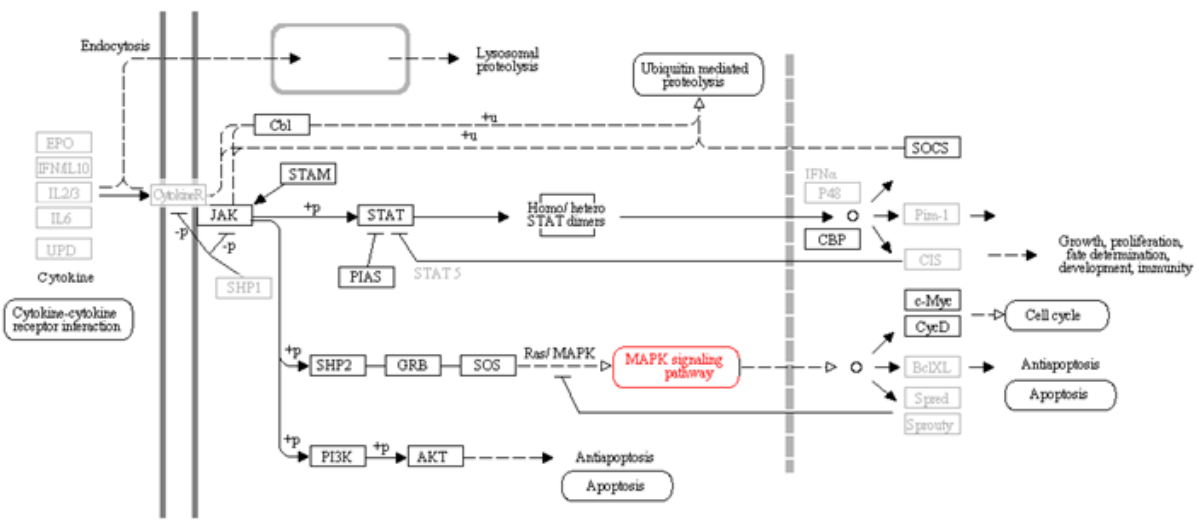


Figura 31 - Via de sinalização JAK-STAT em *Eumetazoa*

Em *Coelomata* mais dois genes são agregados à via, como mostra a Figura 32.

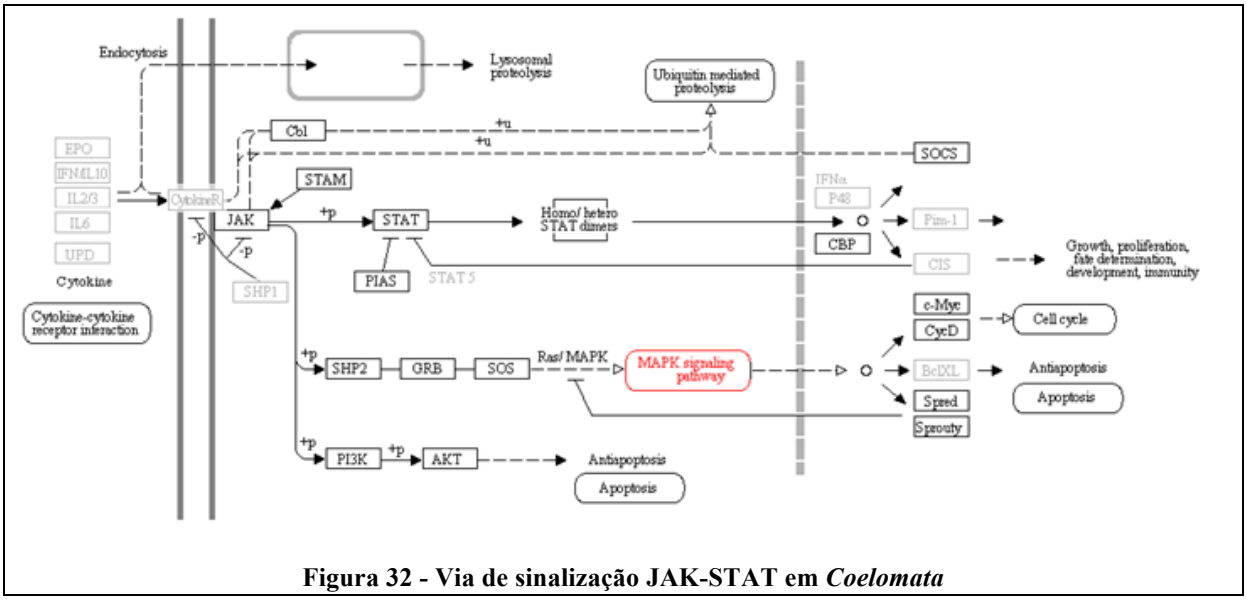


Figura 32 - Via de sinalização JAK-STAT em *Coelomata*

Finalmente, em *Euteleostomi* a via é capaz de receber sinais de fora da célula, provavelmente atingindo a funcionalidade que lhe é atribuída atualmente. A Figura 33 ilustra o gráfico da via, que agora exibe todos os seus genes.

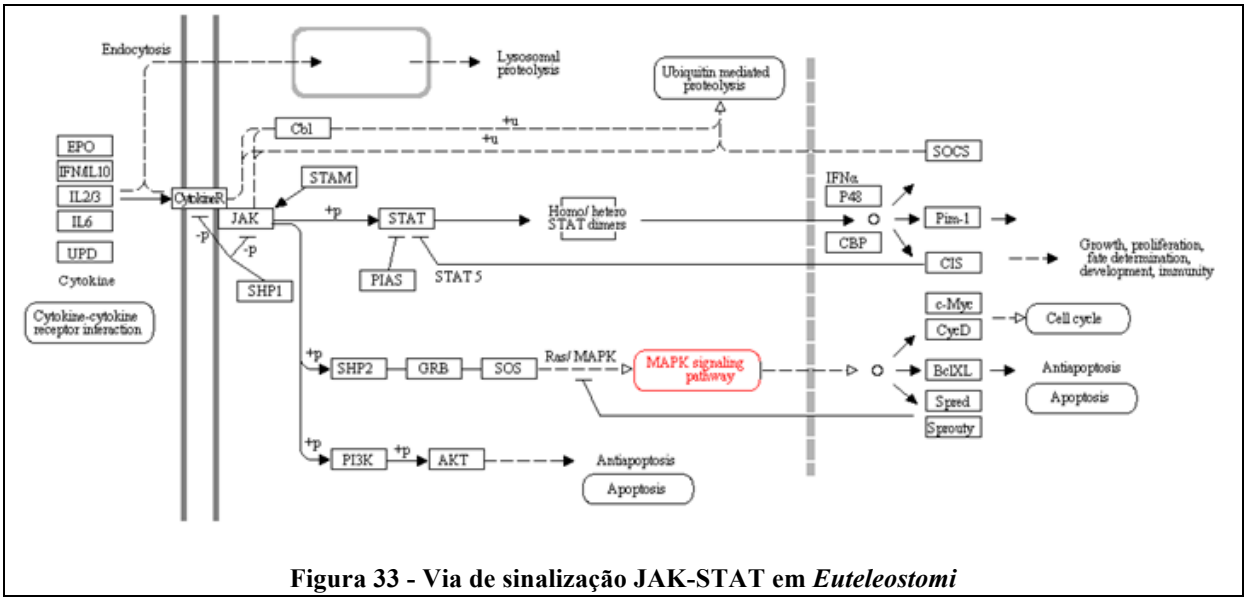


Figura 33 - Via de sinalização JAK-STAT em *Euteleostomi*

4.10 Discussão

As ferramentas apresentadas neste capítulo podem ser de grande valia para pesquisadores interessados em inferir a origem de genes e vias metabólicas e o impacto na evolução da espécie. Um fator chave que está diretamente ligado à precisão dos resultados das análises é a quantidade de organismos representados nos grupos de ortólogos disponíveis.

Espera-se que com o decorrer do tempo, mais espécies sejam sequenciadas e tenham seus genomas totalmente conhecidos. Com isso, que os resultados destas análises tendam a ser cada vez mais precisos. As bases de dados das ferramentas deste trabalho foram construídas de modo a serem facilmente atualizáveis à medida que novos dados estejam disponíveis.

Nos próximo capítulo é apresentado um algoritmo de LCA aperfeiçoado, capaz de detectar a ocorrência de eventos de transferência e perda gênica. Com isso, aumenta a precisão da análise de ancestralidade gênica.

5. LCA Múltiplo e Genesis

O algoritmo de LCA aplicado a grupos de ortólogos descrito no capítulo anterior é bastante preciso na inferência da origem da maioria dos genes. No entanto, esse algoritmo é ineficaz para casos em que genes foram transferidos para organismos em outros ramos da árvore da vida pelo fenômeno conhecido como transferência horizontal (ou lateral). Espera-se que esse evento seja muito mais frequente em genes originados em clados mais recentes de procariotos (como restritos a famílias e transferidos para gêneros distantes). Porém, análises manuais feitas em outros projetos do Laboratório de Biodados detectaram alguns casos em eucariotos. Neste capítulo, é apresentado um novo algoritmo denominado LCA Múltiplo, que além de aumentar a precisão do algoritmo de LCA original, é também capaz de detectar transferências horizontais e deleções gênicas.

A inferência do clado de origem de genes individuais ou de toda uma espécie e a detecção de possíveis transferências laterais é tema de grande interesse da comunidade científica. Com o intuito de fornecer uma ferramenta *online* para inferência da origem gênica de rápido e fácil acesso, foi desenvolvido o *website* Genesis, o segundo produto apresentado neste capítulo. Genesis permite a visualização do resultado do algoritmo de LCA Múltiplo aplicado a um grupo de ortólogos UEKO ou ao conjunto completo de genes de uma dada espécie. O sistema também aceita entradas customizadas, isso é, uma lista de organismos que apresenta um determinado gene e outra com os organismos que não o possuem. O sistema provê resultados em formatos variados, como listas, gráficos e a *Mini LCA Tree*, uma representação gráfica de árvore contendo apenas nós relevantes, que facilitam a visualização e o entendimento.

5.1 LCA Múltiplo

Em alguns dos estudos realizados foram constatados aparentes imprecisões nos resultados das execuções dos algoritmos de LCA. Certos grupos de ortólogos com função específica em vertebrados, por exemplo, tinham LCA resultante em *Eukaryota* devido à presença de uma sequencia relacionada em um protozoário. Uma análise mais aprofundada revelou que os desvios eram causados por possível transferência horizontal de genes. Ao contrário da transferência vertical, na qual os genes são transferidos por hereditariedade, esse é um processo que envolve a transferência de material genético entre organismos não descendentes entre si. Desse modo, é possível que um gene originário em um dado ramo da

árvore da vida seja transferido para outra espécie em um ramo não descendente do primeiro. Nesses casos, o gene original e o transferido continuam apresentando alto grau de similaridade e por isso fazem parte do mesmo grupo de ortólogos. Operacionalmente, isso se resolve se o algoritmo conseguisse particionar a resposta, sugerindo mais de um LCA.

Portanto, para se encontrar o clado de surgimento *de novo* de um gene, é necessário identificar e isolar dentro do grupo de organismos que apresentam ortólogos, aqueles que o adquiriram por transferências horizontais. Esses organismos não devem ser considerados para a inferência do ancestral comum do gene em questão, na espécie que está sendo foco do estudo. O algoritmo utilizado no capítulo anterior não é muito eficaz para a inferência do LCA de genes com casos de transferência horizontal. Isso porque basta que um organismo representado em um grupo de ortólogos UEKO tenha adquirido o gene por transferência horizontal para que o clado de origem resultante da análise de LCA se desloque erroneamente em direção à raiz da árvore.

A Figura 34 ilustra um desvio na origem correta de um grupo de ortólogos causado por transferências horizontal. Nessa imagem, os organismos A, B e C possuem um gene G com surgimento *de novo* no clado representado pelo Nodo 1. Porém, o organismo E também apresenta o gene G, adquirido do organismo C por transferência horizontal. Essa situação faz com que o algoritmo de LCA simples indique erroneamente o Nodo 3 como clado de surgimento do gene G.

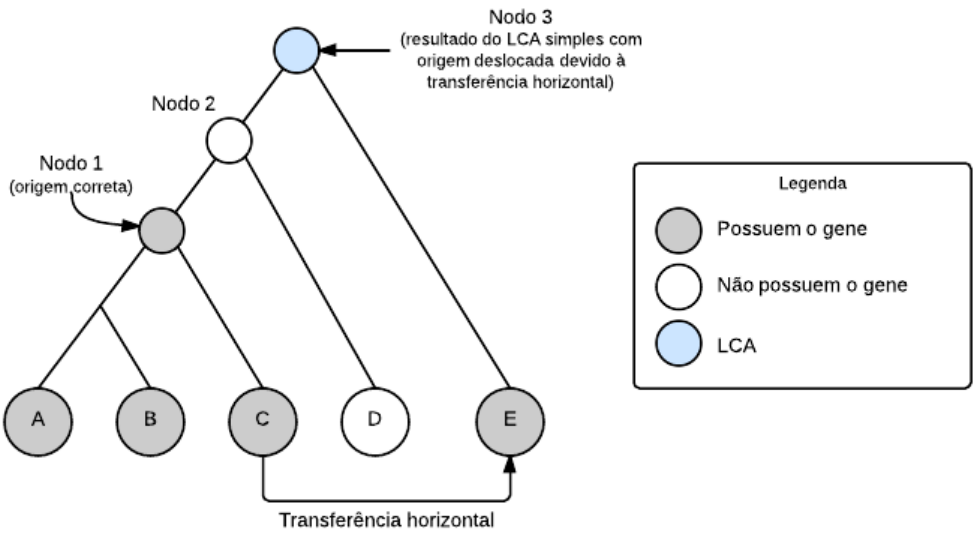


Figura 34 - Deslocamento no LCA original causado pela ocorrência de transferência lateral de genes em um grupo de ortólogos

A fim de corrigir os desvios causados pelas transferências laterais em grupos de ortólogos, o algoritmo de inferência de ancestralidade descrito no capítulo anterior foi aperfeiçoado. O novo algoritmo, denominado LCA Múltiplo, faz uso de uma valiosa informação adicional que auxilia na separação de ortólogos que adquiriram um gene por transferência lateral daqueles que o fizeram pela hereditariedade: a existência de organismos geneticamente mais próximos que comprovadamente não possuem o gene. Portanto, se dentro de um grupo de espécies que apresentam ortólogos há dois subgrupos tão distantes geneticamente entre si a ponto de existirem entre eles organismos que não possuem o gene, é provável que tenha ocorrido transferência lateral de genes entre os ancestrais desses subgrupos. No entanto, também há a possibilidade de perda gênica, que também deve ser avaliada.

Na Figura 34 é possível observar um exemplo da situação descrita acima. Se for sabido que o organismo D comprovadamente não possui o gene G, seu nodo poderia ser utilizado para isolar os organismos que ganharam o gene por hereditariedade (organismos A, B e C) daquele que o adquiriu por transferência horizontal (organismo E). Isso porque D possui um ancestral comum mais próximo de A, B e C (Nodo 2) do que E (Nodo 3). Desse modo, uma das duas situações pode ter ocorrido: um ancestral de D perdeu o gene depois da diferenciação no Nodo 2 ou o organismo E ganhou o gene por transferência horizontal. A escolha entre as duas situações pode não ser uma tarefa simples. No entanto, há heurísticas que podem ser aplicadas. As transferências horizontais geralmente ocorrem entre dois organismos em um evento isolado, portanto, se a maioria dos parentes próximos do organismo E não apresentam o gene, isso é um forte indício de que uma transferência lateral possa ter ocorrido. De modo similar, se os parentes próximos do organismo D possuem o gene G, ganha força a hipótese de deleção gênica.

O objetivo do algoritmo de LCA Múltiplo é, portanto, identificar um ou mais clados, não-descendentes entre si, nos quais um determinado gene apareceu na árvore, seja *de novo,* seja adquirido por transferência. Cada um desses clados constitui um LCA. O algoritmo, porém, não é capaz de diferenciar os eventos de surgimento *de novo* das transferências horizontais. Desse modo, depois de identificados os clados, uma segunda análise deve ser realizada para se verificar a razão do aparecimento do gene. O algoritmo, no entanto, fornece indicativos que podem ajudar nessa análise, tais como o número de descendentes de cada LCA que apresentam o gene e o número de descendentes que comprovadamente não possuem o gene.

O passo-a-passo detalhado de execução do algoritmo de LCA múltiplo, incluindo os materiais e métodos utilizados, pode ser encontrado na seção 3.9. Neste capítulo, é apresentada uma visão geral da lógica do programa e os produtos gerados por ele.

5.1.1 *Visão geral do algoritmo de LCA Múltiplo*

O algoritmo de LCA simples descrito da seção 4.4 calcula o nodo ancestral comum do conjunto de organismos que apresentam um determinado gene. Porém, foi verificado que as transferências horizontais podem fazer com que organismos de ramos não relacionados da árvore possuam um mesmo gene. Para esses casos, o resultado do LCA simples é inválido, pois indica um ancestral comum que nunca possuiu o gene, como ilustra a Figura 34.

O algoritmo de LCA múltiplo tem o objetivo de desmembrar a sub-árvore originada no nodo resultante do LCA simples em sub-árvores menores. As raízes das sub-árvores encontradas pelo algoritmo de LCA múltiplo indicam os clados onde o gene aparece pela primeira vez em um ramo da árvore, seja pelo surgimento *de novo*, seja por transferências horizontais.

A técnica usada para o desmembramento dos LCAs é a intercalação de nodos que comprovadamente **não** apresentam o gene entre os que o possuem. Se um clado que apresenta o gene tem um ancestral direto que não o possui, existe grande probabilidade de o gene ter se originado ali (ou ter sido adquirido por transferência horizontal).

Portanto, a primeira etapa do algoritmo de LCA múltiplo é a classificação de todos os nodos descendentes do LCA simples entre positivos, negativos ou mistos (mistos são os nodos que possuem descendentes viáveis e não viáveis, como será visto posteriormente neste texto). Em seguida, é feita a detecção dos LCAs com base em critérios específicos. Por último, uma árvore simplificada e pronta para exibição visual dos resultados é construída. A tabela a seguir exibe as etapas e os produtos gerados.

Tabela 1 - Etapas e produtos do algoritmo de LCA múltiplo

Etapa	Produto	Descrição do Produto
Construção da *LCA Tree*	*LCA Tree*	Árvore que classifica os nodos como positivos, negativos ou mistos.
Detecção dos Múltiplos LCAs	*LCA Set*	Conjunto de múltiplos LCAs encontrados.
Construção da *Mini LCA Tree*	*Mini LCA Tree*	Árvore de LCAs simplificada, ideal para visualização gráfica

Nas próximas seções, os produtos gerados em cada etapa são descritos com maiores detalhes.

5.1.2 *Lista positiva e negativa*

Neste trabalho foi desenvolvido um *script* para execução do algoritmo de LCA múltiplo em linha de comando. O *script* exige duas listas como entrada: a lista de espécies que possui o gene e a lista de espécies que comprovadamente não possui o gene. A primeira lista, convenientemente chamada de "positiva" neste trabalho, é obtida pela extração dos *taxonomy IDs* do grupo de ortólogos UEKO que contém o gene. A lógica para extração dos *taxonomy IDs* é idêntica àquela descrita no seção 3.6 para o algoritmo de LCA simples.

A segunda lista é obtida pelo método de eliminação. Se a lista de organismos que têm sequencias em um grupo de ortólogos UEKO não possui um determinado organismo que já teve seu genoma totalmente sequenciado, essa espécie provavelmente nunca possuiu o gene ou o perdeu ao longo da evolução. Como os eventos de perda gênica são relativamente raros, a primeira hipótese é mais plausível. Desse modo, de posse da lista de organismos com o genoma totalmente sequenciado, é possível, pelo método de eliminação, a criação de uma lista "negativa" para um determinado gene, isso é, composta por aqueles organismos que não o apresentam. O NCBI disponibiliza uma lista de organismos que já tiveram o genoma totalmente sequenciado, disponível no endereço http://www.ncbi.nlm.nih.gov/genome. Essa foi a lista utilizada neste trabalho.

É importante aqui, no entanto, esclarecer a possibilidade da ocorrência de falsos negativos, e como minimizá-la. Se o grupo de ortólogos UEKO apresentar falsos negativos, ou seja, se um ou mais organismos estiverem incorretamente não representados, esses falso negativos também se propagarão para a lista de negativos criada pelo método de eliminação. Para minimizar essas ocorrências, foi importante neste trabalho manter o sincronismo entre as versões da base UniRef50 utilizada na criação da base UEKO e a lista de genomas completos disponibilizada pelo UniProt. Somente organismos representados na base UniRef50 podem ser utilizados como genomas completos de referência para criação da lista de negativos.

5.1.3 *LCA Tree*

A primeira etapa do algoritmo de LCA múltiplo é a construção de uma nova árvore que servirá como base para a detecção dos múltiplos LCAs. Essa árvore, denominada *LCA*

Tree, é construída a partir do LCA simples da lista positiva recebida pelo *script*, o qual também constitui sua raiz. A partir desse primeiro nodo, o algoritmo de LCA múltiplo constrói o restante da árvore tendo como base o ramo análogo da árvore taxonômica. No entanto, aos nodos da *LCA Tree* são adicionadas informações complementares que serão fundamentais na etapa seguinte do algoritmo.

Basicamente, a construção da *LCA Tree* envolve a seleção dos nodos análogos relevantes da árvore taxonômica e classificação dos mesmos em um dos tipos. Essa classificação é realizada com base na posição do nodo na árvore e nos tipos de seus descendentes diretos.

Um nodo da *LCA Tree* pode assumir um dos tipos abaixo:

- **GENOMA POSITIVO**: O nodo assume esse tipo quando corresponde a um dos organismos da lista de positivos, isso é, quando representa um organismo com genoma conhecido que apresenta o gene (geralmente são nodos-folha da árvore taxonômica, mas nem sempre).

- **GENOMA NEGATIVO**: O nodo assume esse tipo quando corresponde a um dos organismos da lista de negativos, isso é, quando representa um organismo com genoma conhecido que não possui o gene (geralmente são nodos-folha da árvore taxonômica, mas nem sempre).

- **MISTO**: O nodo assume esse tipo quando possui no mínimo um descendente direto positivo (ou genoma positivo) e no mínimo um descendente direto negativo (ou genoma negativo). A presença de descendentes negativos diretos em nodo misto levanta duas hipóteses: os descendentes negativos podem ter perdido o gene ou o nodo misto originou o gene e o transferiu verticalmente a alguns de seus descendentes, o que o configuraria como um dos LCAs.

- **POSITIVO**: O nodo assume esse tipo quando não é um nodo misto e possui pelo menos dois descendentes diretos viáveis. São aqui denominados nodos viáveis aqueles que possuem o gene ou têm capacidade de transferir o gene para seus descendentes, isso é, os positivos, genoma positivos e mistos. Um nodo com essas características é

considerado positivo porque, se possui dois ou mais descendentes diretos viáveis, a maior probabilidade é de que tenha transferido verticalmente o gene para eles.

- **NEGATIVO**: O nodo assume esse tipo quando não é um nodo misto e possui no máximo um descendente direto viável. Um nodo com essas características é considerado negativo, pois se possui apenas um descendente direto viável, a maior probabilidade é de que o gene tenha surgido nesse descendente (o qual será considerado um dos LCAs, como será visto posteriormente).

Um nodo também pode ser considerado alvo de perda gênica. Isso acontece quando um nodo misto possui apenas um genoma descendente negativo. Nesse caso são consideradas duas hipóteses: (i) perda gênica no genoma descendente ou (ii) o genoma descendente é um falso negativo.

A Figura 35 ilustra um exemplo de uma *LCA Tree* para um dado gene Z. Os organismos A, B, C e G possuem o gene Z e por isso seus nodos são classificados com o tipo GENOMA_POSITIVO. Os organismos D, E e F comprovadamente não possuem o gene e por isso seus nodos são classificados com o tipo GENOMA_NEGATIVO. O nodo 1 é classificado como POSITIVO pois todos os seus descendentes diretos são genoma positivos. O nodo 2 é classificado como MISTO porque possui um descendente direto genoma positivo e outro descendente direto genoma negativo. O nodo 3 também é classificado como MISTO pois possui um descendente direto positivo (nodo 1) e um descendente direto genoma negativo (organismo D). O nodo 4 é NEGATIVO, pois possui apenas um descendente direto viável (o nodo 2). Por sua vez, o nodo 5 é classificado como negativo pois também possui apenas um descendente direto viável (o nodo 3).

É fácil notar que no exemplo da Figura 35 que o algoritmo de LCA simples teria originado um resultado impreciso, uma vez que apontaria o LCA resultante para o nodo 5. Como será visto na próxima seção, o algoritmo de LCA múltiplo detecta dois LCAs nesse exemplo: o nodo 1 e o organismo G. Supondo que houve transferência horizontal do gene Z do organismo C para o organismo G, esse seria um resultado satisfatório.

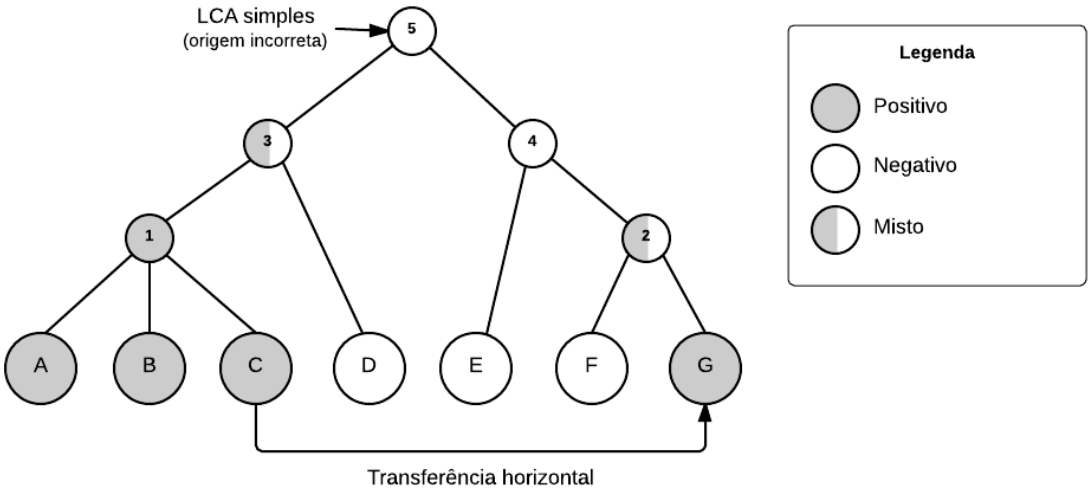


Figura 35 - Exemplo da *LCA Tree*

Além do tipo, são também adicionadas outras informações a cada nodo criado na *LCA Tree.* Essas informações tem o objetivo de auxiliar o pesquisador na posterior diferenciação entre os surgimentos *de novo* das transferências horizontais. Elas são:

- **Número total de descendentes genoma positivos e número total de descendentes genoma negativos**: esses dois números são de extrema importância para a posterior diferenciação dos LCAs que surgiram de transferências laterais daquele que originou o gene *de novo*. Clados ancestrais de um número de genomas negativos consideravelmente maior do que o de genomas positivos têm uma probabilidade maior de terem adquirido o gene por transferência lateral.

- **Nodos de prova positivos e negativos*:*** os nodos de prova são usados posteriormente na construção da *Mini LCA Tree*, que é usada para a representação visual da árvore. Se um nodo foi classificado como positivo, seus nodos de prova associados são dois descendentes genoma positivo. É necessário também que o caminho até cada descendente na árvore seja por meio de um filho direto diferente. Do mesmo modo, para o caso de um nodo classificado como negativo, são associados dois descendentes genoma negativos e, para o caso de nodos mistos, são associados um nodo genoma positivo e um genoma negativo. A utilização de nodos de prova na representação

visual da *Mini LCA Tree* ilustra com maior clareza os motivos de um clado ter sido escolhido LCA.

Os materiais e métodos utilizados para a construção da *LCA Tree* são descritos na seção 3.9.2.

5.1.4 *Detecção dos LCAs*

Depois de construída a LCA Tree, a próxima etapa do algoritmo de LCA múltiplo é a detecção de nodos que são considerados possíveis origens do gene estudado. Esse processo envolve o atravessamento da LCA Tree em busca de nodos que atendam critérios específicos.

Os critérios para caracterização de um clado como LCA são construídos com base na posição do nodo na *LCA Tree*, no seu tipo e nos tipos do nodo pai e dos nodos filhos. Desse modo, um clado é caracterizado como LCA quando seu nodo atende pelo menos um dos três critérios a seguir:

a) O nodo é positivo e possui pai negativo.

Essa situação representa a forma clássica de LCA. Se o nodo é positivo, isso significa que todos os seus descendentes diretos são viáveis, isso é positivos, genoma positivos ou mistos. Se esse nodo possui pai negativo, isso significa que naquele ramo da árvore o gene apareceu a primeira vez ali. Isso permite inferir que: ou o gene surgiu *de novo* no clado correspondente ao nodo ou foi ali adquirido por transferência horizontal, o que caracteriza o nodo como um LCA. A Figura 36 ilustra essa situação. O nodo 2 é positivo pois possui apenas descendentes diretos positivos. Como possui pai negativo, o nodo 2 é caracterizado como LCA.

b) O nodo é o único filho positivo de pai misto.

Se um nodo positivo possui um pai misto, isso significa que também possui irmãos negativos. Porém, se ele é o único positivo entre os irmãos, essa é uma indicação de que seu clado é onde o gene aparece a primeira vez, o que o caracteriza como um LCA. Essa situação é ilustrada na Figura 37. O nodo 2 é misto pois possui descendentes diretos positivos e negativos. Como o único descendente direto positivo é o nodo 3, é maior a probabilidade do gene ter surgido nesse último, o que o caracteriza como um dos LCAs.

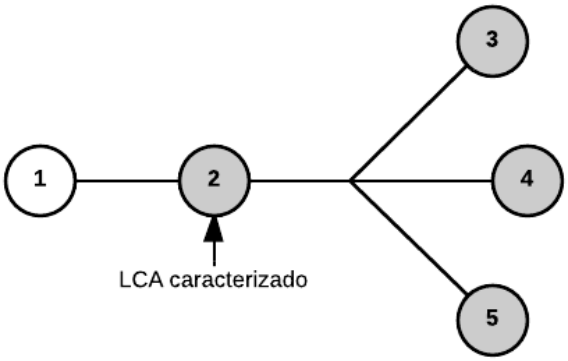


Figura 36 – Primeiro critério para caracterização de LCA: nodo é positivo e possui pai negativo

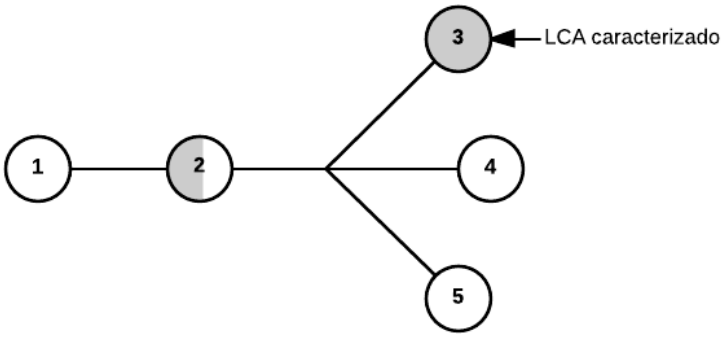


Figura 37 - Segundo critério para caracterização de LCA: nodo é filho único positivo de pai misto

c) O nodo é misto, possui pai negativo e mais de um filho viável.

Se um nodo é misto, ele possui tanto filhos positivos como negativos. Se esse nodo possui pai negativo e pelo menos dois filhos viáveis, há uma forte indicação de que o gene tenha sido gerado no clado representado por ele. Desse modo, esse nodo se torna mais um LCA, como ilustrado na Figura 38. O nodo 2 é misto pois possui dois nodos viáveis (3 e 4) e um não viável (nodo 5). Assim, como possui um pai negativo, ele deve ter originado o gene.

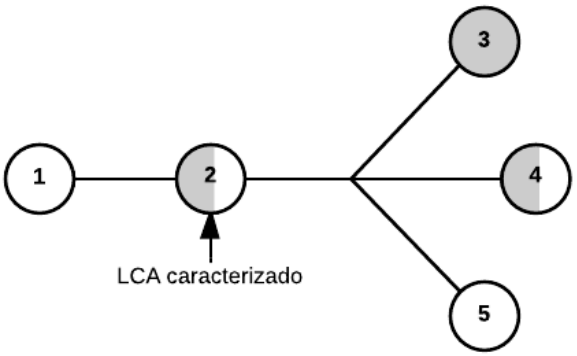


Figura 38 - Terceiro critério para caracterização de LCA: nodo misto com mais de um filho viável

Ao fim do atravessamento da LCA Tree, o resultado é um conjunto de nodos que representam as múltiplas origens de um gene. Cada elemento desse conjunto, além da referência ao clado correspondente ao LCA na árvore taxonômica, também armazena informações adicionais úteis para a posterior diferenciação entre o surgimento de novo das transferências laterais.

A Tabela 2 ilustra um exemplo de resultado do algoritmo de LCA múltiplo executado para o grupo de ortólogos com o identificador K01115, que apresentam o gene da enzima **fosfolipase D1/2 (PLD1_2)**. Nesse caso, foram detectados quatro LCAs. Além do *tax id*, nome do clado e rank, também são exibidos o número de genomas positivos e o número de genomas negativos descendentes de cada LCA. Essa informação pode ser útil para a detecção das transferências horizontais e da origem de novo do gene em estudo. No exemplo citado, a probabilidade maior é de que a enzima tenha se originado em *Eukaryota*, já que esse clado apresenta um número de organismos descendentes que possuem o gene consideravelmente maior que os demais. Portanto, os clados *Rhodobacteraceae*, *Burkholderiales* e *Mycobacterium* devem ter adquirido o gene por transferência horizontal, provindo de organismos eucariotos.

Os materiais e métodos utilizados para a detecção dos múltiplos LCAs são detalhados na seção 3.9.3.

Tabela 2 - Exemplo de lista de LCAs encontrados para o gene da enzima PLD1_2

#	Txid	Clado do LCA	Rank	Nº genomas positivos	Nº genomas negativos
1	31989	*Rhodobacteraceae*	família	9	11
2	80840	*Burkholderiales*	ordem	8	68
3	1763	*Mycobacterium*	gênero	18	22
4	2759	*Eukaryota*	super-reino	289	21

Muitas ferramentas estão disponíveis *online* e *off-line* para a representação visual de árvores. A maioria delas é capaz de ler formatos universais, tais como *Newick* [42], *Nexus* [43], *phyloXML* [44] e JSON [45]. A *LCA Tree* gerada no segundo passo do algoritmo pode ser facilmente convertida para algum desses formatos. Porém, por ser uma árvore auxiliar gerada apenas como base para a detecção dos LCAs, a *LCA Tree* poderá conter até centenas de nodos. Essa complexidade inviabiliza a utilização da mesma para a representação visual dos resultados, uma vez que seria extremamente grande e confusa. Para resolver esse problema, o último passo do algoritmo gera uma versão compacta da *LCA Tree*, a qual foi denominada *Mini LCA Tree*.

A *Mini LCA Tree* mantém apenas nodos essenciais para a boa visualização dos resultados. Um nodo é considerado essencial se:

a) Representa um dos LCAs.
b) Algum de seus filhos diretos é LCA.
c) Corresponde a um dos elementos da lista dos clados que devem sempre ser exibidos. Neste trabalho são sempre exibidos os clados *cellular organisms*, *Eukaryota*, *Bacteria* e *Archaea*, por serem clados de referência.

Na *Mini LCA Tree* também são adicionados os chamados *nodos de prova*. Esses nodos são úteis na representação gráfica de uma árvore de LCAs, pois permitem a rápida visualização do motivo que levou um determinado nodo a ser qualificado como positivo, negativo ou misto.

A Figura 39 exibe um setor da *Mini LCA Tree* resultante da execução do algoritmo de LCA múltiplo para o grupo de ortólogos KO com o identificador K00226, a enzima **diidroorotato desidrogenase (DHODH)**. É possível observar na figura que *Mycoplasma* é um nodo misto, uma vez que possui descendentes diretos positivos e negativos. No entanto, esse clado é o único descendente direto viável de seu nodo pai (*Mycoplasmataceae),* como é

possível verificar devido à presença de um nodo de prova negativo (*Ureaplasma urealyticum)*. Isso faz com que *Mycoplasma* seja caracterizado como um dos LCAs. Além disso, é possível notar que o super-reino *Bacteria* também se trata de um nodo misto devido a dois nodos de prova positivos incluídos na árvore.

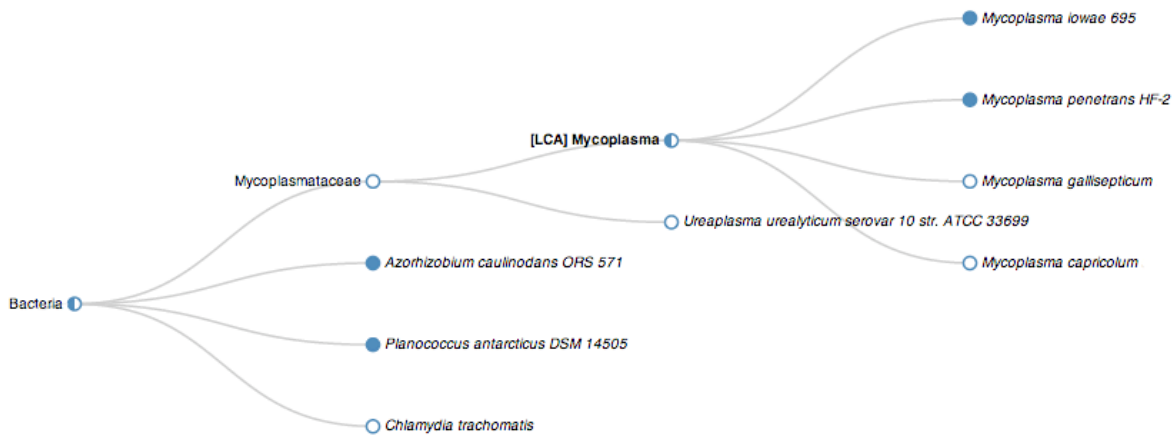


Figura 39 - Setor da *Mini LCA Tree* construída para o grupo de ortólogos K00226

Nota-se, portanto, que a *Mini LCA Tree* é uma árvore explicativa, criada com o intuito de esclarecer os motivos que levaram às escolhas dos LCAs.

Os materiais e métodos utilizados para a construção da *Mini LCA Tree* são descritos na seção 3.9.4.

5.2 Genesis

O algoritmo de LCA Múltiplo pode ser uma ferramenta de grande valia para os pesquisadores. Além de inferir com precisão o clado de origem de genes, servindo como ponto de partida para estudos de ancestralidade gênica, o programa também dá ótimas pistas sobre a ocorrência de transferências horizontais de genes entre espécies. As transferências horizontais constituem assunto de grande interesse da comunidade científica, pois constituem um forte mecanismo que molda o processo evolutivo.

O *script* desenvolvido para o LCA Múltiplo, no entanto, foi construído para execução em linha de comando e exige a prévia configuração das bases de dados necessárias na máquina local. Desse modo, a utilização do programa para grandes volumes de dados tende a se tornar demorada. Além disso, programas que executam por linha de comando possuem

interface de entrada de dados pouco amigável e não fornecem uma boa representação visual dos resultados.

De modo a contornar as dificuldades citadas, foi também desenvolvido neste trabalho o *website* Genesis. Esse sítio, disponível gratuitamente para toda a comunidade científica, permite que pesquisadores realizem seus próprios estudos sobre a origem múltipla de genes utilizando interface gráfica amigável e intuitiva. Os resultados são exibidos graficamente e possuem opção de *download* em diferentes formatos.

Genesis é um *website* totalmente responsivo, isso é, sua interface se adapta de acordo com a resolução de tela onde é exibido. Desse modo, o pesquisador poderá acessá-lo não apenas a partir de seu *desktop*, mas também de *smartphones* e *tablets*.

O *website* Genesis, acessível em http://biodados.icb.ufmg.br:8080/Genesis, permite que o usuário escolha entre três tipos de ferramenta para análise de ancestralidade gênica. Cada ferramenta tem um diferente foco de análise, a qual pode ser realizada para genes individuais, para uma dada espécie ou baseada em uma entrada personalizada. A tela principal do *site* apresenta o *menu* de opções, onde o usuário pode optar por uma das ferramentas. A Figura 40 ilustra a tela principal do *website* Genesis.

Como é possível observar na Figura 40, o website Genesis disponibiliza três ferramentas *online* principais:

a) ***KO Origin* (origem dos genes):** Realiza a análise de ancestralidade de um grupo de ortólogos KO específico. Os resultados são os múltiplos LCAs do conjunto de organismos apresentados pelo grupo de ortólogos KO.

b) ***Species Origin* (origem das espécies)**: Realiza a análise completa de ancestralidade de uma única espécie. Os resultados exibem o número de genes que surgiram em cada clado na linhagem evolutiva da espécie. Além disso, o *site* exibe um gráfico de linhas para análise visual do resultado. Também lista os genes que surgiram por possíveis transferências laterais.

c) ***Multi LCA Tree* (árvore de múltipla origem)**: Esse tipo de análise permite que o usuário entre com uma lista personalizada de organismos que possuem um determinado gene e uma lista dos quem não o possuem. O sistema efetua o algoritmo de LCA Múltiplo em tempo real e exibe os LCAs em uma lista. A *Mini LCA Tree* também é exibida graficamente. Esta aplicação permite que o usuário

entre com a ocorrência taxonômica de ortólogos determinados por quaisquer *software*.

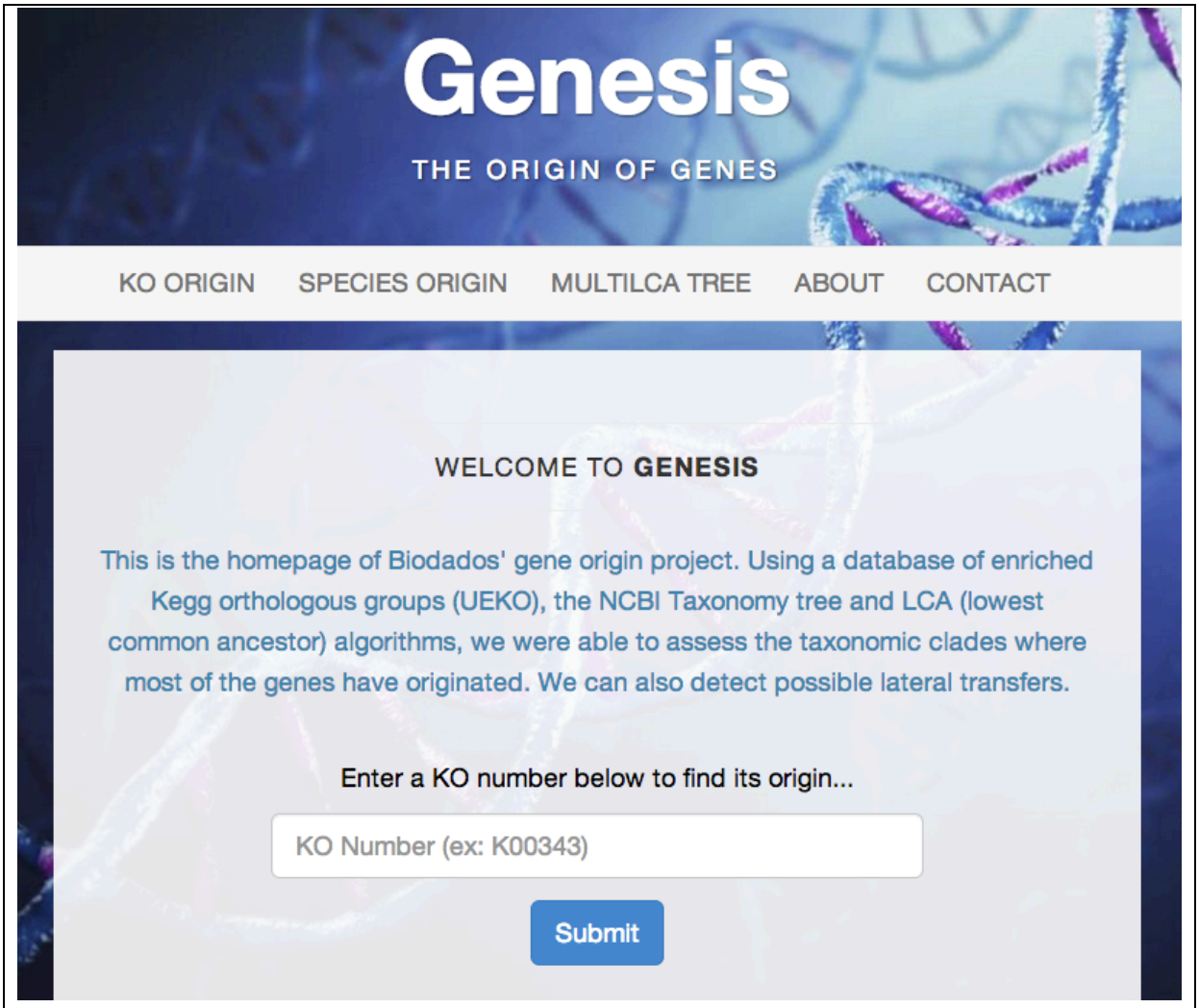


Figura 40 - Tela principal do *website* Genesis

Cada uma das ferramentas *online* é apresentada com mais detalhes nas próximas seções. A seção 3.10 descreve com detalhes os materiais e método utilizados para o desenvolvimento do *website* Genesis.

5.2.1 *KO Origin (Origem dos genes)*

A primeira ferramenta *online* do *website* Genesis provê ao usuário uma interface visual que permite rápido acesso aos múltiplos LCAs de um determinado gene, representado por seu grupo de ortólogos KO.

Para consultar os múltiplos LCAs de um grupo de ortólogos KO, o usuário deve digitar o identificador do mesmo no campo apropriado, como ilustrado na Figura 41.

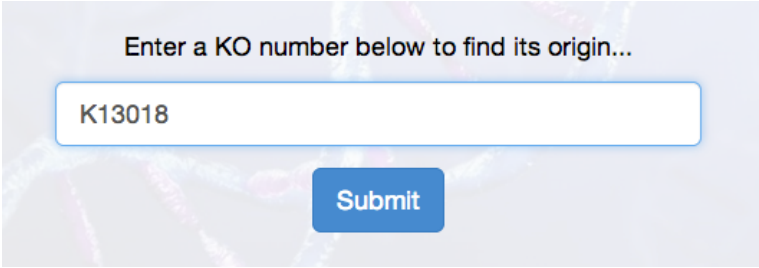


Figura 41 - Campo para entrada do identificador de um grupo de ortólogos KO em Genesis

Para essa funcionalidade os resultados são pré-computados, isso é, uma rotina executa o *script* de LCA múltiplo para todos os grupos KO existentes e deposita os resultados em um banco de dados. Quando o usuário clica no botão *Submit*, o sistema consulta nesse banco de dados os múltiplos LCAs do grupo KO correspondente ao identificador e os exibe em uma lista, como ilustrado na Figura 42.

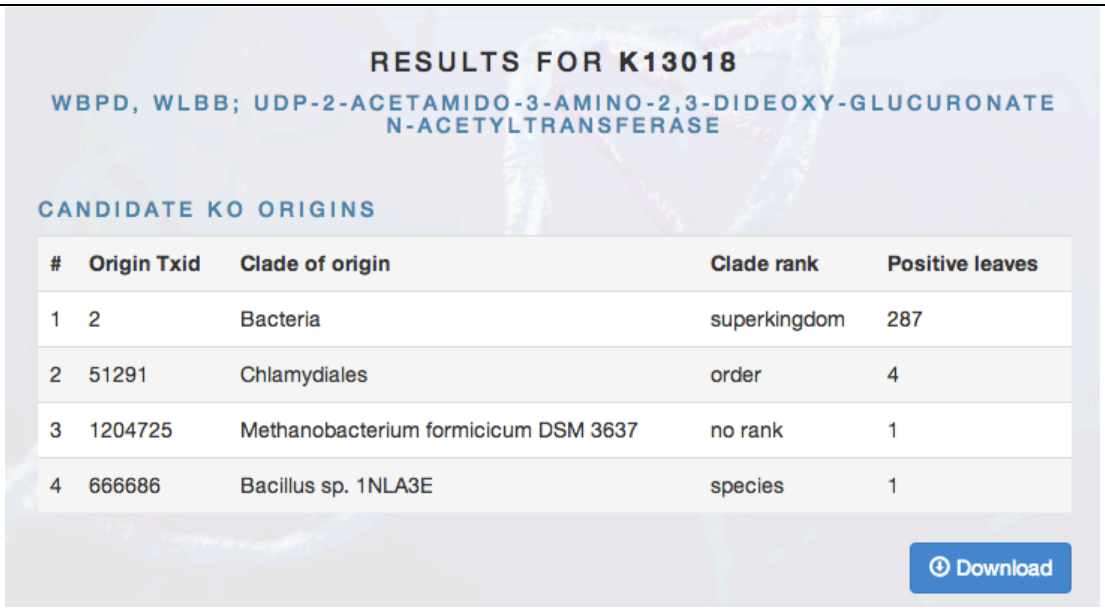


Figura 42 – Exemplo de resultado no *website* Genesis exibindo múltipas origens de um gene

Cada linha da lista representa um dos LCAs encontrados, isto é, uma possível origem do gene pesquisado. As colunas apresentadas na lista são as seguintes:

#: Sequencial do LCA

Origin Txid: O NCBI *Taxonomy* ID do clado do LCA.

Clade of origin: Nome científico do clado do LCA.

Clade rank: *Rank* do clado do LCA.

Positive leaves: O número de genomas positivos descendentes, isto é, o número de organismos descendentes do clado que apresentam o gene.

Como já foi explicitado anteriormente, apenas um dos LCAs encontrados representa a provável origem *de novo* do gene enquanto os demais são candidatos a terem o adquirido por transferências horizontal. No entanto, o número de genomas positivos descendentes pode ser um bom indicativo da origem *de novo* do gene. Um LCA que possui um número de genomas positivos descendentes muito superior aos demais tem maior probabilidade de representar a verdadeira origem do gene. Na Figura 42, por exemplo, é provável que o clado de origem *de novo* tenha sido *Bacteria*, enquanto os demais o adquiriram por transferência horizontal.

O pesquisador, se desejar, também poderá efetuar o *download* dos resultados em formato TSV (colunas separadas por tabulações). Para isso, basta clicar no botão *Download*, exibido na Figura 42 logo abaixo da lista.

5.2.2 *Species origin (origem das espécies)*

A segunda ferramenta *online* do *website* Genesis permite ao usuário realizar análises de ancestralidade em nível de espécie. Para isso, o usuário deverá acessar o menu *Species Origin* e entrar com o *NCBI Taxonomy ID* do organismos desejado, como demonstrado na Figura 43.

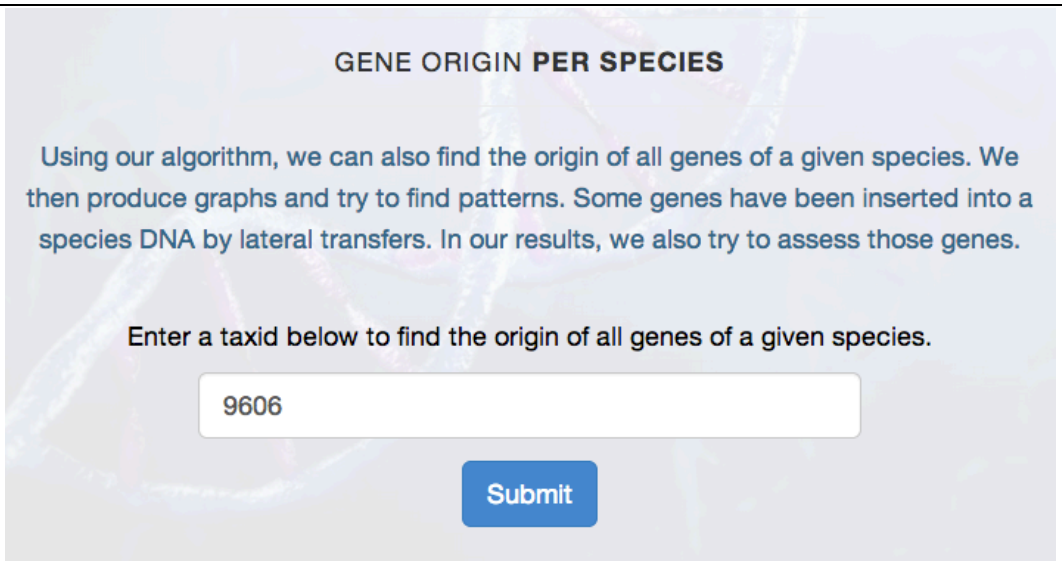


Figura 43 - Tela do *website* Genesis para consulta da origem de todos os genes de uma espécie

A análise de ancestralidade de uma espécie em Genesis produz uma série de quatro resultados de interesse. São eles:

1. a lista completa dos genes da espécie pesquisada acompanhados de seu clado de origem.
2. os genes que foram adquiridos por possíveis transferências laterais.
3. a contagem de surgimentos de genes em cada clado na linhagem da espécie pesquisada.
4. um gráfico do resultado do item 3.

Da mesma forma que para a consulta por genes individuais, os resultados também são obtidos de um banco de dados pré-computado.

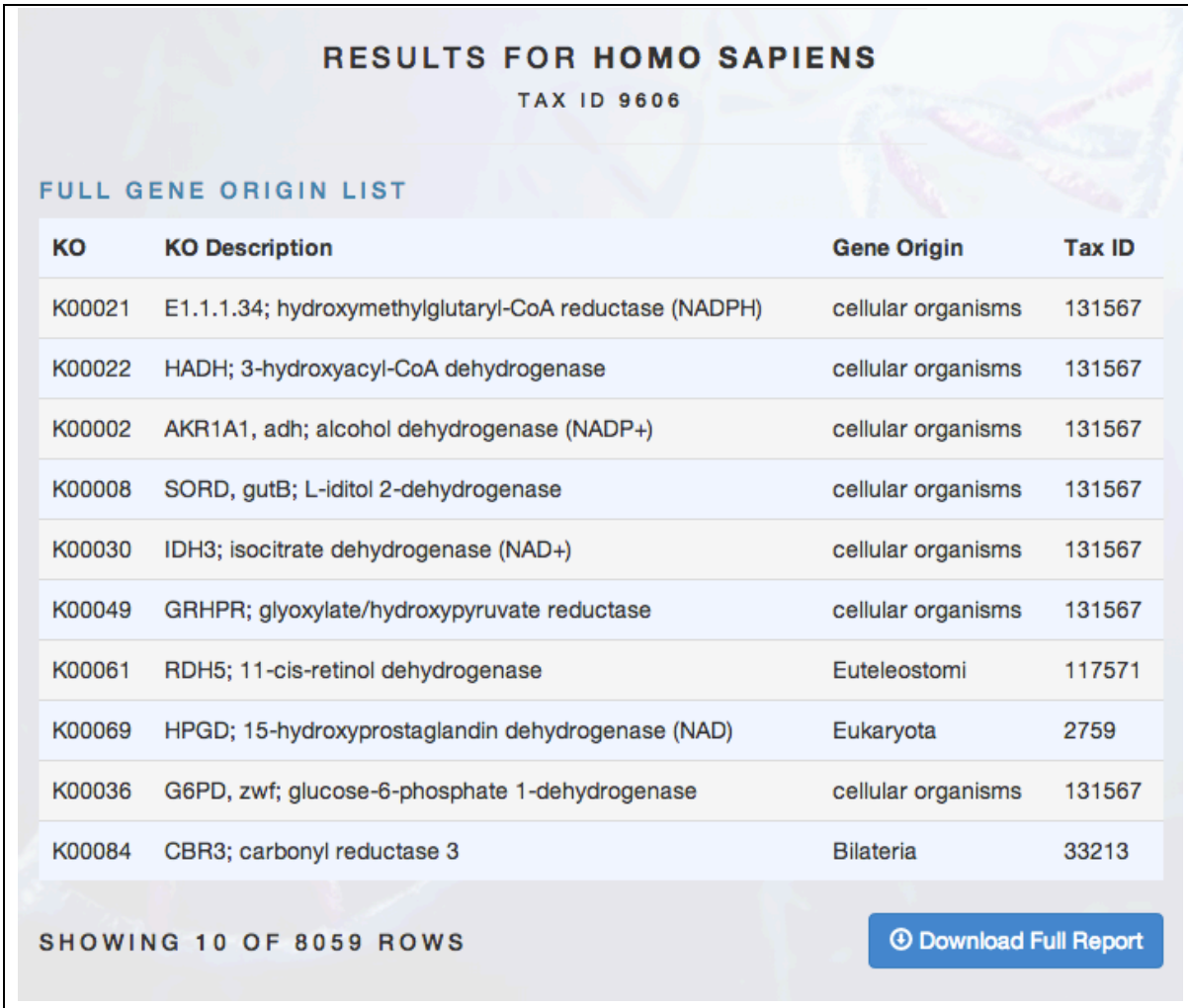
Para se chegar aos resultados acima, o sistema deve também verificar, para cada gene transcrito pela espécie, se o seu surgimento ocorreu *de novo* ou se foi adquirido por transferência horizontal. Para isso, para cada gene verifica-se entre os LCAs múltiplos encontrados se aquele com o maior número de genomas positivos descendentes corresponde a um dos clados da linhagem da espécie pesquisada. Se isso acontece, é provável que o gene representado pelo KO tenha tido origem *de novo* em um dos clados da linhagem da espécie pesquisada. Caso contrário, o gene pode ter sido adquirido por transferência horizontal.

São geradas, portanto, duas listas: uma de grupos de ortólogos KO que tiveram origem na linhagem da espécie pesquisada e outra de genes que foram adquiridos por transferências horizontais. Essas listas são exibidas nos resultados da análise, representando, respectivamente, os itens a) e b) listados acima. A Figura 44 e a Figura 45 ilustram exemplos desses resultados.

As colunas apresentadas nos relatórios são as seguintes:

KO: O identificador do grupo de ortólogos KO que representa o gene.
KO Description: A descrição do KO (geralmente nome da enzima codificada).
Gene Origin: O clado de origem encontrado para o gene.
Tax ID: O *NCBI Taxonomy ID* do clado de origem.

É possível observar nos exemplos acima que apenas os dez primeiros itens das listas resultantes são exibidos na tela. Para obter a lista completa de genes e suas origens, o usuário deve clicar no botão *Download Full Report.*



RESULTS FOR HOMO SAPIENS

TAX ID 9606

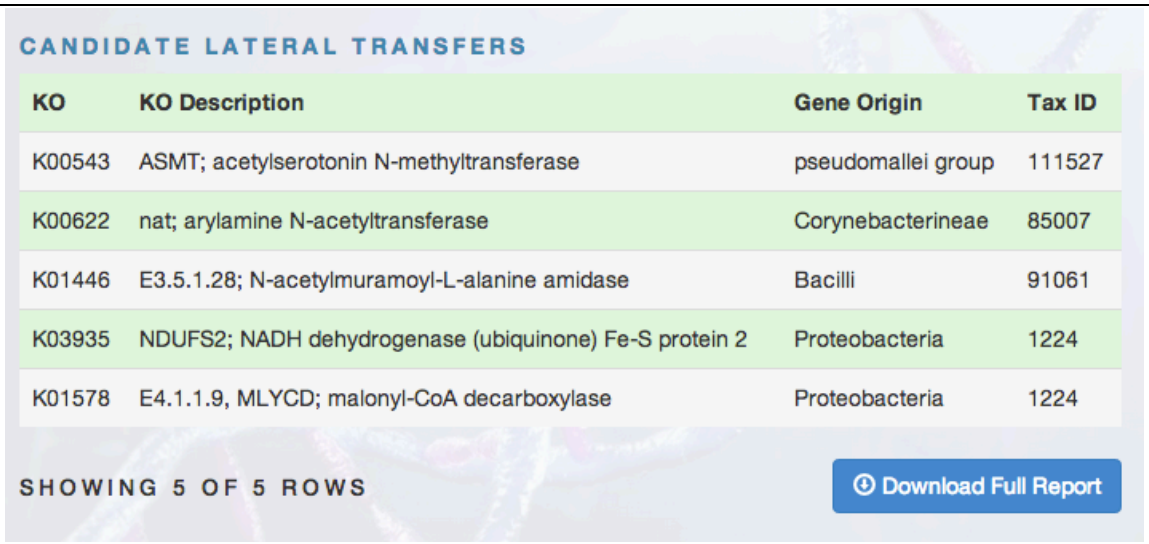
FULL GENE ORIGIN LIST

KO	KO Description	Gene Origin	Tax ID
K00021	E1.1.1.34; hydroxymethylglutaryl-CoA reductase (NADPH)	cellular organisms	131567
K00022	HADH; 3-hydroxyacyl-CoA dehydrogenase	cellular organisms	131567
K00002	AKR1A1, adh; alcohol dehydrogenase (NADP+)	cellular organisms	131567
K00008	SORD, gutB; L-iditol 2-dehydrogenase	cellular organisms	131567
K00030	IDH3; isocitrate dehydrogenase (NAD+)	cellular organisms	131567
K00049	GRHPR; glyoxylate/hydroxypyruvate reductase	cellular organisms	131567
K00061	RDH5; 11-cis-retinol dehydrogenase	Euteleostomi	117571
K00069	HPGD; 15-hydroxyprostaglandin dehydrogenase (NAD)	Eukaryota	2759
K00036	G6PD, zwf; glucose-6-phosphate 1-dehydrogenase	cellular organisms	131567
K00084	CBR3; carbonyl reductase 3	Bilateria	33213

SHOWING 10 OF 8059 ROWS

Download Full Report

Figura 44 - Relatório do *website* Genesis exibindo as origens de todos os genes de uma espécie



CANDIDATE LATERAL TRANSFERS

KO	KO Description	Gene Origin	Tax ID
K00543	ASMT; acetylserotonin N-methyltransferase	pseudomallei group	111527
K00622	nat; arylamine N-acetyltransferase	Corynebacterineae	85007
K01446	E3.5.1.28; N-acetylmuramoyl-L-alanine amidase	Bacilli	91061
K03935	NDUFS2; NADH dehydrogenase (ubiquinone) Fe-S protein 2	Proteobacteria	1224
K01578	E4.1.1.9, MLYCD; malonyl-CoA decarboxylase	Proteobacteria	1224

SHOWING 5 OF 5 ROWS

Download Full Report

Figura 45 - Relatório do *website* Genesis exibindo genes de uma espécie possivelmente adquiridos por transferência horizontal

O resultado seguinte da análise de ancestralidade de espécies constitui na sumarização da quantidade de genes surgidos em cada clado da linhagem. Para isso, o sistema efetua a contagem dos genes que se originaram em cada clado na linhagem da espécie pesquisada. Não são contabilizados aqueles adquiridos por transferência horizontal.

Os resultados mais uma vez sugerem um padrão de picos de surgimento de genes em determinados clados. No exemplo demonstrado na Figura 46 é possível observar o gráfico gerado pelo *website* Genesis para o surgimento de genes em *Homo sapiens.*

A contagem de surgimento *de novo* de genes por clado também é apresentada em formato de lista com opção de *download*. A Figura 47 mostra um trecho de um exemplo desse relatório.

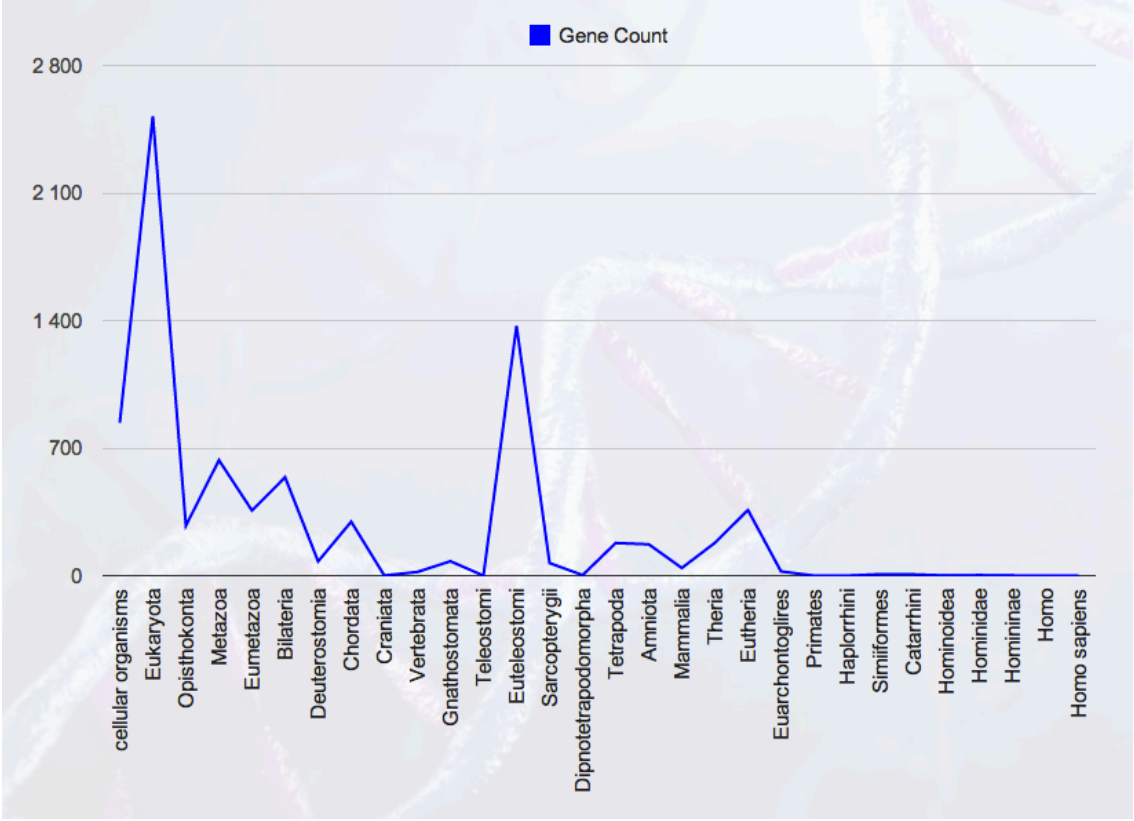
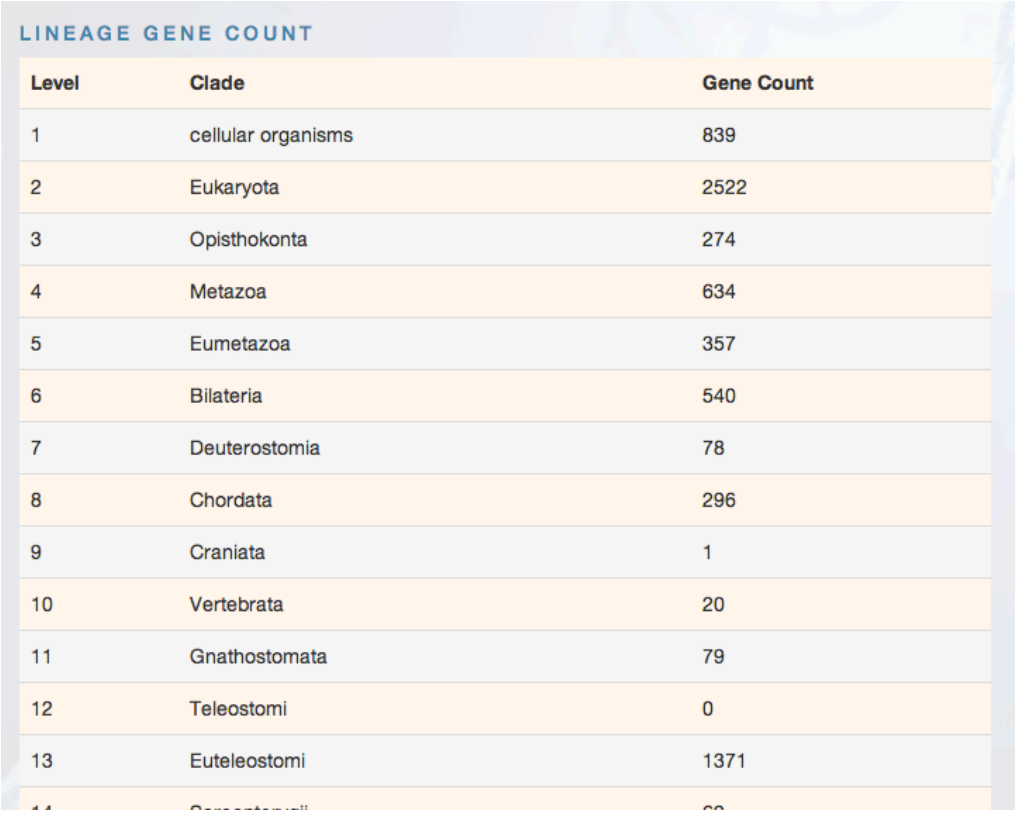


Figura 46 - Gráfico gerado pelo *website* Genesis para o surgimento de genes em *Homo sapiens*



LINEAGE GENE COUNT

Level	Clade	Gene Count
1	cellular organisms	839
2	Eukaryota	2522
3	Opisthokonta	274
4	Metazoa	634
5	Eumetazoa	357
6	Bilateria	540
7	Deuterostomia	78
8	Chordata	296
9	Craniata	1
10	Vertebrata	20
11	Gnathostomata	79
12	Teleostomi	0
13	Euteleostomi	1371

Figura 47 - Relatório do *website* Genesis exibindo a contagem do surgimento de genes humanos por clado

5.2.3 *Árvore de múltipla origem*

Os grupos de ortólogos KO utilizados neste trabalho são construídos por curadoria manual pelos bioinformatas da KEGG e incrementados com sequências da base UniRef50 pelo algoritmo UEKO. Este trabalho mostrou que os grupos de ortólogos criados pela KEGG podem ser utilizados para a inferência do clado de origem de muitos genes. No entanto, para genes pouco estudados ou recém-descobertos, a escassez de sequencias pode resultar em grupos de ortólogos pouco representativos. Tal fator pode ser determinante para a precisão do LCA resultante da análise de ancestralidade. Além disso, o processo de anotação funcional utilizado pelos principais centros de bioinformática, por ser muitas vezes feito de forma automática, está sempre suscetível a erros e desvios.

Desse modo, a terceira ferramenta do *website* Genesis é talvez a de maior relevância para bioinformatas interessados na ancestralidade de genes recém-descobertos ou com literatura escassa. Diferentemente das duas ferramentas anteriores, onde as consultas são realizadas em bases pré-computadas, a ferramenta Multi LCA Tree permite que o pesquisador insira um conjunto personalizado de organismos de entrada para a realização do LCA

múltiplo em tempo real. Desse modo, o cientista pode realizar análises de ancestralidade a partir de resultados de suas próprias pesquisas de homologia gênica.

A ferramenta também possibilita que uma lista de seja carregada de um dos grupos de ortólogos KO existente e eventualmente modificada. Essa funcionalidade permite que o pesquisador remova os falsos positivos e/ou falsos negativos que detectar nas listas pré-existentes e realize uma nova análise de LCA Múltiplo, a qual pode resultar em LCAs mais precisos que os pré-computados.

A Figura 48 ilustra a tela de entrada de dados da ferramenta *Multi LCA Tree*. Como é possível observar na figura, a tela apresenta três campos editáveis e uma caixa de seleção, os quais são descritos a seguir:

- **Campo *Positive organisms***: A lista de organismos positivos deve ser preenchida com os NCBI *tax ids* das espécies que possuem o gene. Cada *tax id* deve ser colocado em uma linha da caixa de texto, como ilustrado na Figura 48.

- **Campo *Negative Organisms***: De modo similar, a lista de organismos negativos deve ser preenchida com os *tax ids* das espécies que comprovadamente não expressam o gene. Cada *tax id* deve ser colocado em uma linha da caixa de texto, como ilustrado na Figura 48.

- **Campo *Load organisms from KO*:** Este campo opcional permite o preenchimento das duas listas anteriores com *tax ids* provindos de um grupo existente de ortólogos KO. Essa funcionalidade possibilita que o pesquisador corrija falsos positivos ou falsos negativos presentes em grupos de ortólogos KO existentes. Dessa forma, uma nova análise pode produzir melhores resultados. Para carregar listas de organismos pré-existentes, o usuário deve preencher este campo com o identificador do KO e clicar no botão *Load.*

- **Caixa de seleção *Consider negative all complete genomes not presente in the positive list***: Quando essa caixa é selecionada, o sistema considera como negativos todos os organismos com genoma completo que não estejam presentes na lista positiva. Essa funcionalidade é útil quando o pesquisador dispõe apenas dos *tax ids*

dos organismos que expressam um gene e deseja inferir suas origens ou buscar por transferências horizontais.

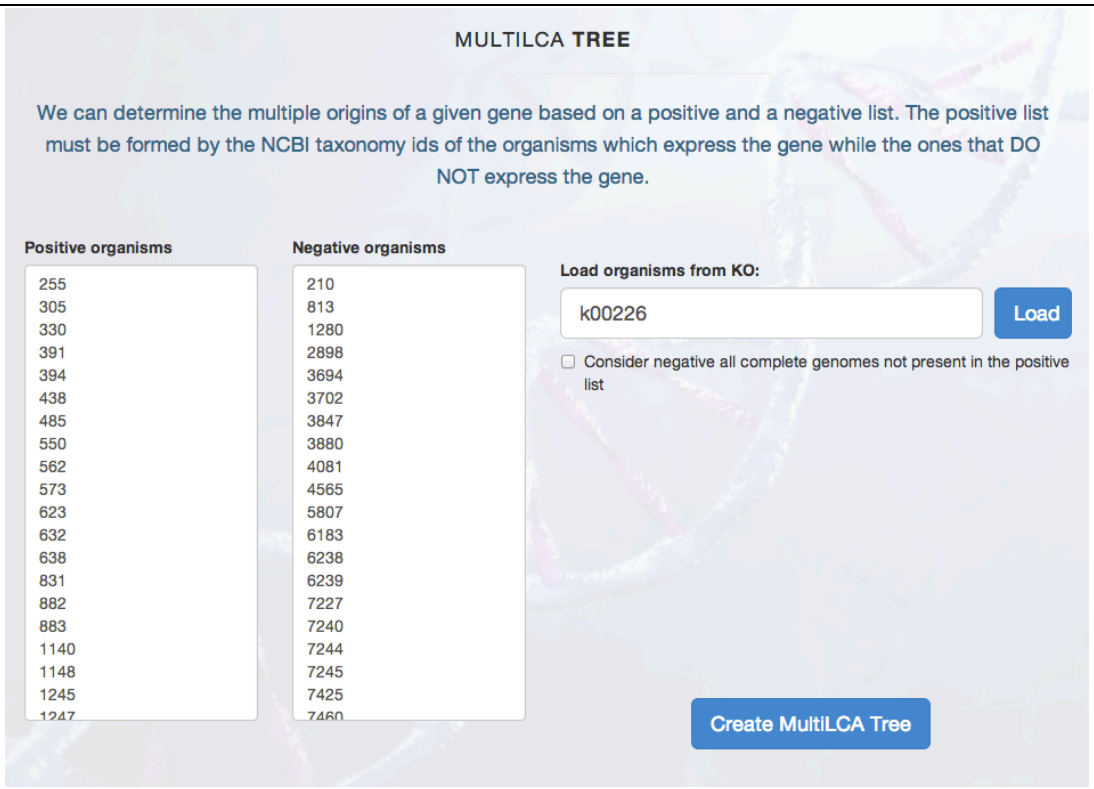


Figura 48 - Tela de entrada de dados para a *MultiLCA Tree* no *website* Genesis

Após do preenchimento correto dos campos descritos acima, o usuário deve clicar no botão *Create MultiLCA Tree.* Essa ação dispara o algoritmo de LCA múltiplo no sistema que recebe como entrada as listas de *tax ids* presentes dos campos. A execução do algoritmo pode levar de 40 segundos a 1 minuto. Como descrito na seção 5.1, esse algoritmo produz um conjunto de múltiplos LCAs resultantes e uma *Mini LCA Tree.* Esses dados são utilizados para a exibição visual dos resultados no *website* Genesis.

Os materiais e métodos utilizados para a execução do algoritmo de LCA múltiplo em Genesis e para a exibição visual da *Mini LCA Tree* são descritos na seção 3.10.3.

A Figura 49 exibe um exemplo de relatório gerado pelo *website* Genesis contendo a lista de LCAs encontrados para um determinado gene. O relatório é uma tabela em que cada linha representa um dos LCAs encontrados. As colunas são descritas a seguir:

#: Sequencial do LCA

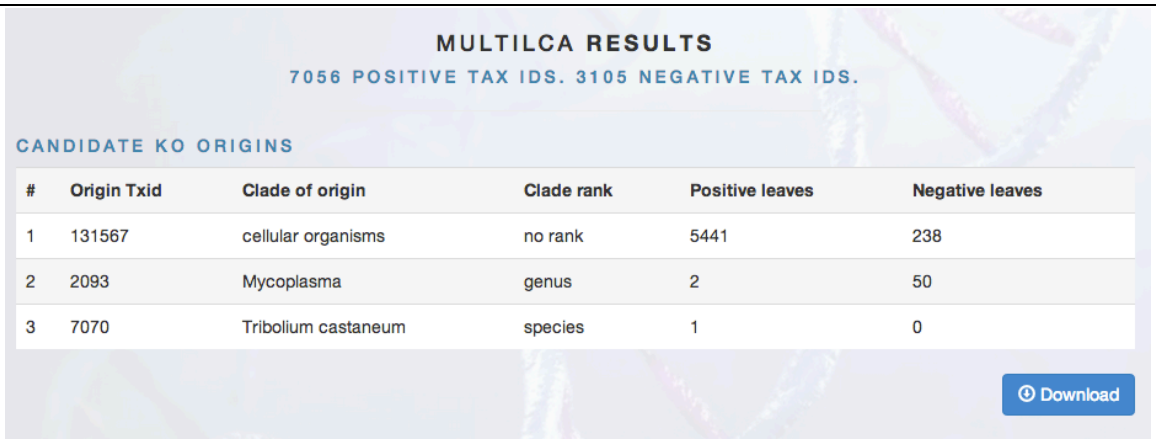
Origin Txid: O NCBI *Taxonomy* ID do clado do LCA.

Clade of origin: Nome científico do clado do LCA.

Clade rank: *Rank* do clado do LCA.

Positive leaves: O número de genomas positivos descendentes, isto é, o número de organismos descendentes do clado que apresentam o gene.

Negative leaves: O número de genomas negativos descendentes, isto é, o número de organismos descendentes do clado que não apresentam o gene.



MULTILCA RESULTS

7056 POSITIVE TAX IDS. 3105 NEGATIVE TAX IDS.

CANDIDATE KO ORIGINS

#	Origin Txid	Clade of origin	Clade rank	Positive leaves	Negative leaves
1	131567	cellular organisms	no rank	5441	238
2	2093	Mycoplasma	genus	2	50
3	7070	Tribolium castaneum	species	1	0

Download

Figura 49 - Relatório do *website* Genesis exibindo os LCAs encontrados em uma análise

Além dessas informações, o relatório também traz outros dados relevantes em seu cabeçalho: o número de organismos positivos e o número organismos negativos considerados na análise.

O pesquisador pode ainda fazer o *download* dos resultados em formato TSV (colunas separadas por tabulações) ao clicar no botão *Download.*

Além da lista de LCAs demonstrada acima, o sistema também exibe a *Mini LCA Tree* gerada pelo algoritmo de LCA Múltiplo. Para isso, a árvore é convertida em formato JSON e a biblioteca *D3.js* é utilizada (ver em métodos).

A Figura 50 e a Figura 51, que fazem parte dos estudos de caso de trabalho, exibem *Mini LCA Trees* geradas no *website* Genesis. Os nodos positivos e genoma positivos são preenchidos com a cor azul. Os nodos negativos e genoma negativos não são preenchidos. Já os nodos mistos são preenchidos com uma cruz azul. Os LCAs encontrados são destacados em negrito e também pelo prefixo [LCA].

É fácil notar como a visualização da árvore favorece o entendimento dos resultados. É possível encontrar na árvore a posição e os ancestrais mais conhecidos de cada LCA encontrado, assim como os nodos de prova.

5.3 Estudos de Caso

A fim de se verificar a eficácia do *website Genesis* e o algoritmo de LCA Múltiplo na detecção das múltiplas origens de um gene, foram realizados dois estudos de caso com exemplos reais.

O primeiro estudo foi realizado com a proteína *Janus kinase 1* (JAK1). Essa proteína é uma tirosina cinase envolvida na sinalização da resposta a *interferons.* A origem da via de resposta a *interferon* está sendo estudada no Laboratório de Biodados da UFMG pela mestranda Verônica Melo. Utilizando-se a proteína humana como semente do programa *SeedServer* (Rafael L. M. Guedes e colaboradores, não publicado), foi criado um grupo de homólogos e o *Taxonomy Id* das proteínas agrupadas foi recuperado.

Embora o LCA Simples indicasse a origem do grupo de ortólogos em *Opisthokonta,* por análise manual suspeitava-se que a sequencia do organismo *Monosiga ovata* teria sido agrupada como artefato ou adquirida por transferência horizontal. Todavia, executando-se o algoritmo de LCA Múltiplo por meio do *website* Genesis, obtêm-se a árvore exibida na Figura 50.

Figura 50 - Árvore obtida pelo algoritmo de LCA Múltiplo executado sobre o grupo de ortólogos de JAK1

Pela análise da figura, percebe-se que o organismo *Monosita ovata* foi isolado em novo LCA. Por isso, não há suporte para uma conexão vertical entre esse organismo e os demais 25. A análise sugere que a origem do grupo de homólogos é *Chordata,* com suporte de 25 organismos como indicado à direita desse clado na árvore. O organismo *Schizosaccharomyces pombe 972h-* foi incluído como nodo de prova negativo utilizado para

separar *Monosida ovata* dos demais. O organismo *Xenopus laevis* aparece para dar suporte positivo ao LCA em *Chordata.*

O segundo estudo de caso realizado envolve análise da ancestralidade em genes de organismos procariotos. Segundo a base de dados *MicrobesOnLine* [46], o gene *marR (Multiple antibiotic resistance protein)* de *E. coli* é restrito à família *Enterobacteriaceae*. Essa base considera um amplo conjunto de informações, incluindo sintenia e regiões regulatórias, para determinar grupos de ortólogos. Informa, portanto, que há herança vertical do gene restrita ao clado *Enterobacteriaceae.* Por outro lado, o gene *marR* pertence ao grupo K03712 na base *Kegg Orthology.* O grupo K03712 inclui bactérias de várias outras famílias. Como esperado, o algoritmo de LCA Simples aplicado a esse grupo de ortólogos retorna *Bacteria* como ancestral.

Os resultados da Figura 51 mostram que o LCA Múltiplo soluciona esse problema. Ele aponta um LCA em *Enterobacteriaceae* e sugere que ocorre uma transferência gênica entre esse grupo e *Clostridiales*. Na árvore aparece ainda um terceiro LCA em *Enterobacteriaceae bacterium strain FGI 57.* Ele aparece dentro do grupo *Unclassified Enterobacteriaceae* que possui quatro descendentes negativos e um positivo, o que justifica a presença do LCA de acordo com o algoritmo. No entanto, trata-se de um grupo de organismos não classificados, portanto o mesmo não resistiria a uma inspeção manual.

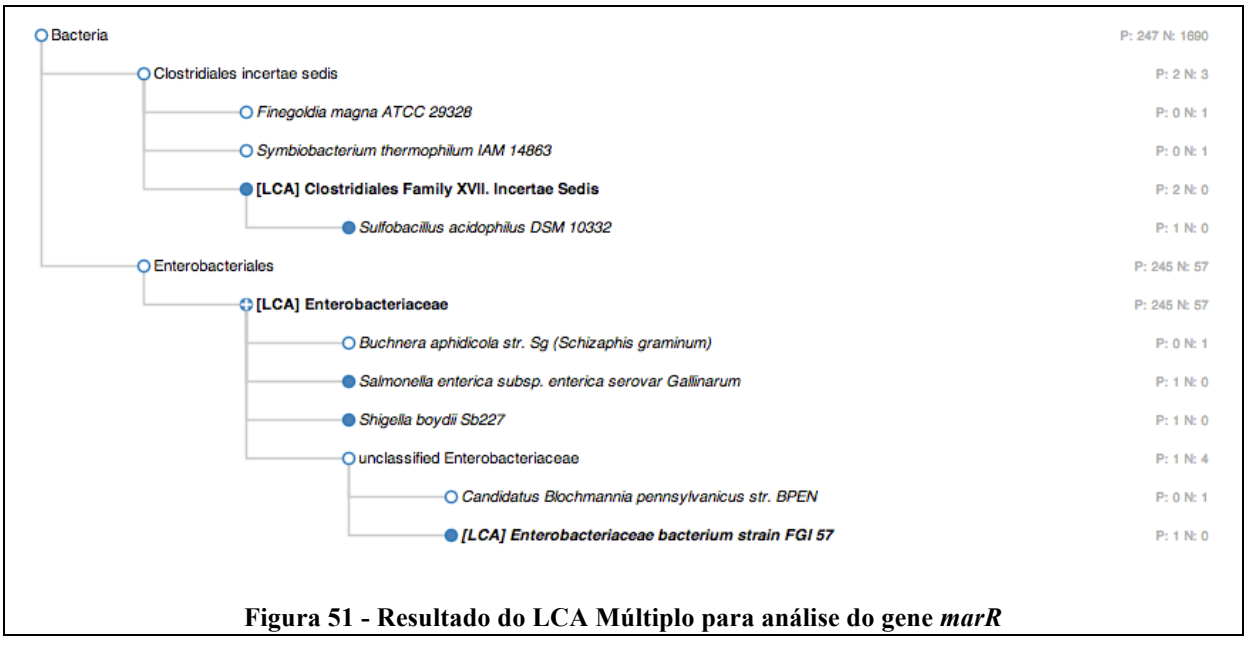


Figura 51 - Resultado do LCA Múltiplo para análise do gene *marR*

Portanto, a origem do grupo de ortólogos que envolve o gene *marR* de *E. Coli* é restrita a *Enterobacteriaceae*, o que corrobora com a classificação feita pelo *MicrobesOnLine.*

5.4 Discussão

O algoritmo de LCA Múltiplo diminui o grave problema do LCA Simples que ocorre com mais frequência em procariotos, mas também em eucariotos, ocasionado pelo fenômeno das transferências laterais de genes. Além disso, ele protege a análise contra sequencias similares, de clados distantes, que possam ter sido agrupadas erroneamente em grupos de ortólogos. O LCA Simples levaria a origem a um nodo muito mais próximo da raiz. Consequentemente os gráficos de origem dos genes ao longo da linhagem evolutiva de um organismo tornam-se mais acurados.

É a primeira vez que se pode verificar graficamente os períodos de aparecimento de genes de qualquer espécie, e isso é prontamente ofertado pelo *website* Genesis. A inspeção do perfil de aparecimento de genes em *Homo sapiens* (9606), *Bos taurus* (9913), *Gallus gallus* (9031), *Latimeria chalumnae* (7897), *Danio rerio* (7955) e *Branchiostoma floridae* (7739) suportam a possibilidade de que os organismos mais complexos tenham adquirido conjuntos de genes adicionais subsequentemente, e observações desse tipo são possíveis somente com a disponibilização imediata dos resultados no *website*.

O algoritmo de LCA múltiplo, no entanto, é muito dependente da lista de genomas completos. Ao se utilizar uma lista recente, é importante que as sequencias desses organismos estejam disponíveis nos agrupamentos de ortólogos. Assim, ressalta-se a importância da metodologia de enriquecimento do UEKO, pois organismos ainda não trabalhados pela KEGG seriam incorporados. A ausência do ortólogo de um organismo com genoma completo recentemente sequenciado criaria um falso negativo, extremamente danoso ao funcionamento do algoritmo.

Para melhorar o algoritmo seria necessário contar com filogenia, onde a hierarquia da árvore seria muito mais exata. Trabalhar com a árvore taxonômica do NCBI impõe esse limite. O que alivia um pouco o problema é a criação constante de grupos “no rank” pelo NCBI, que repartem alguns grupos taxonômicos, adicionando um pequeno grau de filogenia.

O acoplamento do algoritmo com *softwares* de reconstrução filogenética parece uma perspectiva interessante. Por exemplo, nos casos em que se tenha 4 nodos negativos e um positivo, quando estudada a filogenia, poderia se perceber que todos são descendentes de um

ancestral comum filho do nodo estudado. Os nodos mistos analisados com filogenia poderiam ser até mesmo eliminados, pois se saberia exatamente onde ocorreu a especiação.

No entanto, apesar as limitações citadas, o algoritmo de LCA múltiplo e o *website* Genesis já constituem ferramentas valiosas para pesquisadores interessados na ancestralidade de genes. A ferramenta de *Multi LCA Tree* de *Genesis* permite, por exemplo, que listas positivas e negativas customizadas sejam utilizadas. Desse modo, o pesquisador pode progressivamente excluir os falsos negativos e falsos positivos das análises a fim de obter resultados cada vez mais preciso.

6. BOWS

Ferramentas computacionais poderosas são necessárias para lidar com a crescente quantidade de dados gerados pelos projetos de sequenciamento de DNA e com os novos padrões expostos pelas abordagens sistêmicas. Biólogos e bioinformatas são amplamente beneficiados com programas de alto desempenho em suas pesquisas, porém, muitas vezes enfrentam uma ampla gama de dificuldades para tê-los funcionando e produzindo resultados.

Primeiramente, os pesquisadores muitas vezes necessitam lidar com complexos procedimentos de instalação, uma vez que as diferentes ferramentas dependem de bibliotecas de sistema específicas. A instalação de bibliotecas de sistema nem sempre é um processo simples já que muitas delas possuem vários graus de dependências além de possuírem compatibilidade limitada a apenas um ou poucos sistemas operacionais. Uma vez que muitas das ferramentas de Bioinformática são construídas para executar exclusivamente em um sistema operacional, muitas vezes os bioinformatas são forçados a se familiarizar com diferentes sistemas e versões.

Outra dificuldade encontrada é que algumas ferramentas demandam altíssimo poder de processamento, recurso que muitas vezes não está disponível localmente para o pesquisador. Computadores de alto desempenho são dispositivos caros, o que frequentemente os torna inacessíveis para laboratórios menores.

Por último, os pesquisadores ainda esbarram na curva de aprendizado de cada ferramenta de interesse. Programas de Bioinformática geralmente são executados em linha de comando e sua configuração adequada depende da familiarização com os parâmetros de entrada e os formatos de saída de cada um.

Os bioinformatas teriam um grande ganho se fosse possível utilizar um único sistema que lhes desse acesso à maioria das ferramentas que utilizam em seu dia-a-dia, com uma interface comum de entrada de parâmetros e respostas em formato conhecido. Os *Web Services,* devido à sua natureza universal e interface amplamente conhecida, constituem a alternativa ideal para prover essa solução. Além disso *Web services* disponibilizam uma camada segura que pode restringir o acesso a servidores e ferramentas, além de proteger o material que possui direitos autorais.

Neste capítulo é apresentado BOWS, uma plataforma de serviços remotos que utiliza a tecnologia de *Web services* para prover acesso assíncrono e centralizado e uma interface comum de entrada e saída a um conjunto heterogêneo de ferramentas de Bioinformática. Por meio dos *Web services* acessíveis em seu *front-end,* BOWS provê acesso seguro e protege os

direitos autorais a ferramentas que podem executar em *clusters* de alto desempenho. Modelado de forma genérica, BOWS suporta virtualmente qualquer ferramenta de bioinformática em seu *back-end*. Além disso, pela natureza universal dos *Web services,* pode ser chamado a partir de qualquer linguagem de programação com suporte à tecnologia SOAP.

BOWS é distribuído gratuitamente como um aplicativo Web *stand-alone,* em formato WAR, e dessa maneira pode ser instalado em qualquer centro de bioinformática. Isso abre um novo caminho para que esses centros distribuam ferramentas próprias, bases de dados e resultados de um modo fácil e seguro.

6.1 Funcionamento de BOWS

A plataforma BOWS funciona intermediando o acesso entre o programa do usuário e o servidor onde a ferramenta desejada está instalada. Dessa forma, BOWS é instalado em um servidor intermediário, de onde expõe dois *Web services*: um de *front-end*, direcionado para os usuários de ferramentas, e outro de *back-end* (ou *admin),* utilizado pelos administradores das ferramentas. Por meio do *front-end*, o programa do usuário pode:

a. submeter novos processos para execução assíncrona em uma das ferramentas registradas na plataforma;
b. checar o *status* de um processo submetido;
c. obter os resultados de um processo finalizado.

Para cada uma das operações listadas acima, BOWS disponibiliza um método remoto.

O *Web service* de *back-end* (ou *admin)* consiste de métodos remotos que devem ser usados pelos proprietários das ferramentas a serem registradas em BOWS. A função principal do *back-end* é enviar respostas às requisições assíncronas realizadas pelos usuários. As seguintes operações são realizáveis por meio do *back-end:*

a. busca por novos processos na fila de execução de uma ferramenta;
b. alteração do *status* de um processo;
c. inserção dos resultados de um processo na plataforma;

Além disso, o *back-end* pode ser utilizado para o registro de uma nova ferramenta na plataforma e para a exclusão de uma ferramenta registrada. As chamadas ao *back-end* de

BOWS são geralmente realizadas a partir do servidor onde as ferramentas são executadas, dentro de um *script* denominado *arrow*. Esse *script,* que deve ser executado periodicamente, tem a função de verificar novos processos, executar a ferramenta e inserir os resultados da execução na plataforma BOWS. O formato do *script arrow* é explicado com maiores detalhes na seção 6.4.

A arquitetura de funcionamento da plataforma é exibida na Figura 52.

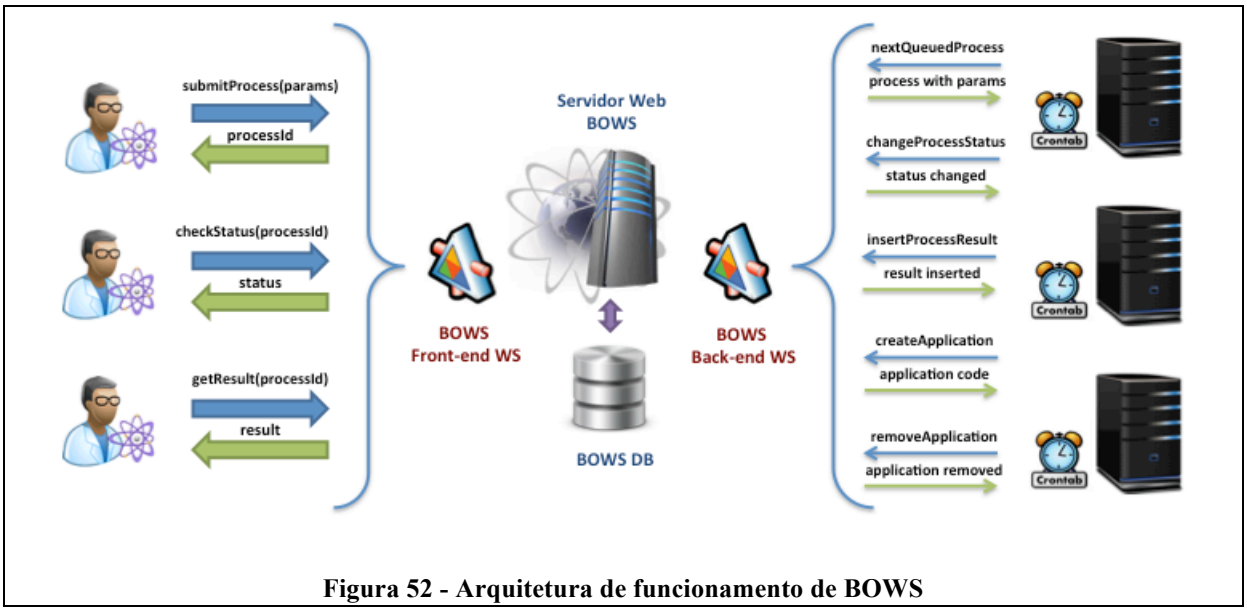


Figura 52 - Arquitetura de funcionamento de BOWS

Como é possível observar na Figura 52, o servidor *Web* de BOWS expõe os dois *Web services.* Por meio do serviço de *front-end*, pesquisadores podem submeter processos, checar *status* e obter resultados. Do outro lado do diagrama estão os servidores onde as ferramentas são executadas. Esses servidores não possuem portas abertas – executam periodicamente o *script arrow*, o qual chama o *back-end* de BOWS em busca de novos processos. Quando um processo é encontrado e executado, o serviço de *back-end* é chamado novamente para inserção dos resultados. Esses servidores que acessam BOWS pelos serviços de *back-end* seriam, idealmente, hipercomputadores (*clusters* computacionais).

6.2 Ciclo de BOWS

O ciclo de BOWS se inicia quando um processo é submetido via *front-end* pelo programa do bioinformata a uma das ferramentas registradas na plataforma BOWS e termina quando o resultado da execução do processo é retornado. A Figura 53 exibe um diagrama de sequência na linguagem UML que ilustra o ciclo de funcionamento da plataforma BOWS.

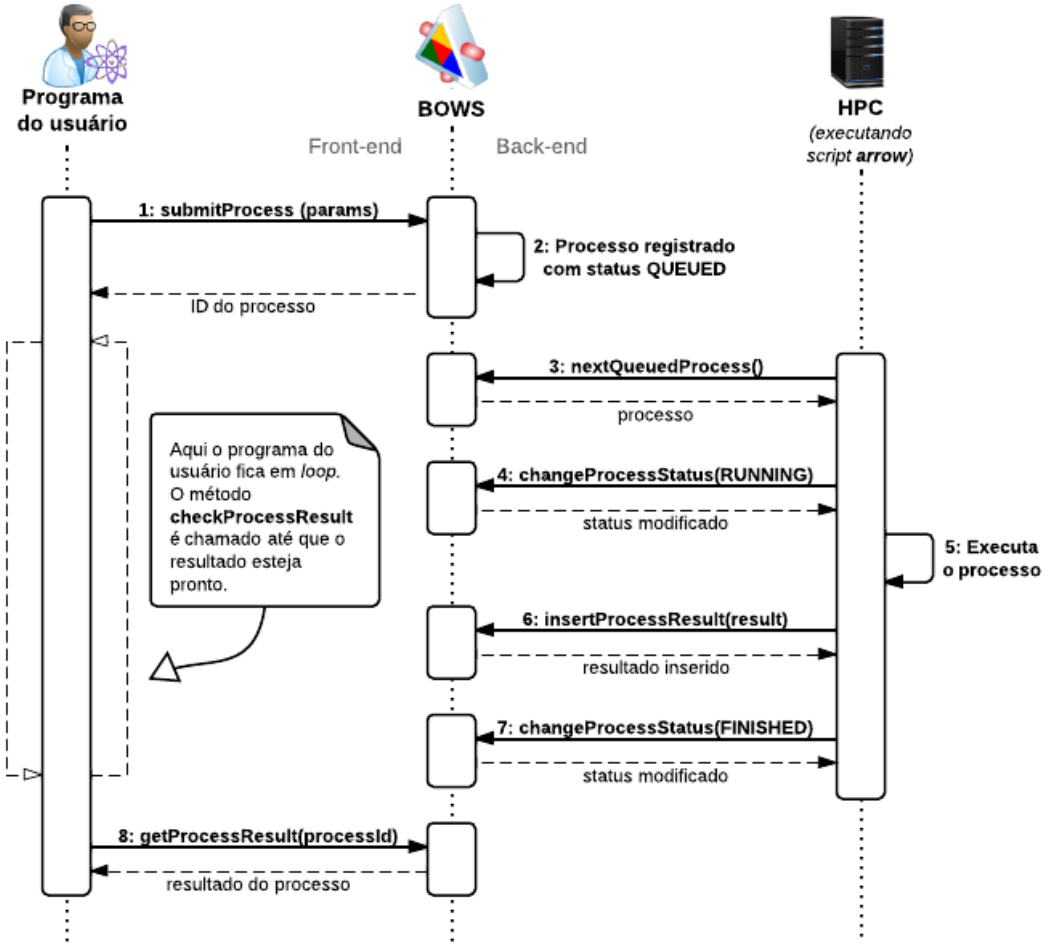


Figura 53 - Diagrama de sequência da plataforma BOWS

Como é possível observar na Figura 53, o ciclo de BOWS tem inicio quando o programa do usuário submete um novo processo por meio do *front-end* da plataforma. Para isso, o método *submitProcess* deve ser chamado com o código do aplicativo a ser executado e uma lista de parâmetros para execução. O novo processo é então criado no banco de dados de BOWS com o *status* QUEUED (enfileirado) e um identificador é retornado para o programa do usuário. Nesse momento, o programa do usuário entra em um *loop* de execução no qual a cada iteração o método *checkProcessResult* do *front*-end de BOWS é chamado. O programa permanece em *loop* enquanto o *status* do processo seja diferente de FINISHED (terminado) ou ERROR (erro).

Enquanto isso, no servidor que hospeda a ferramenta alvo, o *script arrow* é executado periodicamente em busca de novos processos na plataforma BOWS. Para isso, *arrow* verifica se há novos processos na fila de BOWS utilizando o método *nextQueuedProcess* do *back-end.*

O primeiro processo na fila de execução é retornado. Nesse momento o *script arrow* modifica o *status* desse processo para RUNNING (em execução) por meio do método do *changeProcessStatus*. O *script arrow* então executa a ferramenta alvo utilizando os parâmetros definidos para o processo. Ao fim da execução, o resultado é gravado novamente no servidor de BOWS por meio de chamada ao método *insertProcessResult* do *back-end*. Em seguida o *status* do processo é modificado para FINISHED com uma nova chamada ao método *changeProcessStatus*.

Uma vez que o *status* do processo submetido é modificado para FINISHED, o programa do usuário pode enfim sair de seu *loop* de execução. O resultado do processo é lido por meio do método *getProcessResult* do *front-end* de BOWS e seus dados estarão enfim disponíveis para uso no restante do programa.

6.3 Transações de *Front-end*

O serviço de *front-end* da plataforma BOWS provê métodos remotos que possibilitam ao programa do usuário submeter processos a ferramentas registradas e ler os resultados. Como se trata de respostas assíncronas, tipicamente o programa do usuário deverá entrar em *loop* aguardando a disponibilidade do resultado de um processo. De forma geral, o programa do usuário deverá operar da seguinte forma para acessar uma ferramenta da plataforma BOWS:

1. Submeter um novo processo com os parâmetros desejados.
2. Entrar em *loop*, sendo que a cada iteração deverá verificar o *status* do processo submetido.
    a. Se o *status* for igual FINISHED ou ERROR, deverá sair do *loop.*
3. Caso o *status* do processo seja FINISHED, deverá chamar novamente o *front-end* para obter os resultados.
4. Caso o *status* do processo seja ERROR, deverá tratar o erro.

O serviço de *front-end* da plataforma BOWS disponibiliza três métodos, como demonstrado na Tabela 3. O cliente para o *front -end* de BOWS deve ser gerado por meio de seu documento WSDL. Na instalação padrão da plataforma, é possível acessar esse documento pelo caminho `http://<servidor>:8080/services/BOWS?wsdl`.

Tabela 3 - Métodos de *front-end* da plataforma BOWS

Método	Descrição
submitProcess	Submete um novo processo para uma ferramenta registrada na plataforma BOWS.
checkProcessResult	Verifica o *status* de um processo.
getProcessResult	Obtém o resultado de um processo.

O método *submitProcess* deve ser utilizado para a submissão de novos processos a uma ferramenta registrada na plataforma BOWS. Cada novo processo deve ser submetido juntamente com os parâmetros de execução. O retorno será o identificador do processo criado, o qual deve ser utilizado posteriormente para verificação do *status* de execução e obtenção dos resultados. A Tabela 4 exibe os parâmetros aceitos pelo método *submitProcess* e suas descrições.

Tabela 4 - Parâmetros do método *submitProcess*

Parâmetro	Descrição
applicationCode	O código da ferramenta em que se deseja submeter um processo.
submissionParam[]	Uma lista de objetos do tipo *submissionParam* (parâmetros de execução).

Como é possível notar na Tabela 4, o segundo parâmetro do método *submitProcess* é uma lista de objetos do tipo *SubmissionParam.* Cada objeto dessa lista deve apresentar o nome e o valor de um dos parâmetros de execução da ferramenta, além de outras informações. Os valores dos parâmetros devem estar em conformidade com as regras estabelecidas no momento do registro da ferramenta. Por exemplo, se um parâmetro foi registrado com um *matcher*, seu valor deve ser compatível com a expressão regular definida. Além disso, todos os parâmetros obrigatórios devem estar presentes. A Tabela 5 exibe os campos do tipo *SubmissionParam* e suas descrições.

Tabela 5 - Campo do tipo *SubmissionParam*

Nome do campo	Descrição
name	O nome do parâmetro
textValue	O valor textual do parâmetro (só deve ser preenchido caso *binary* seja *false)*
binaryValue	O valor binário do parâmetro (só deve ser preenchido caso *binary* seja *true)*
mimeType	O tipo MIME do valor.
binary	Deve-se passar *true* caso o valor seja binário ou *false* caso seja textual.

Pode-se notar pela análise da tabela que o usuário poderá passar tanto valores textuais como binários para os parâmetros (por binários, entendem-se arquivos compactados do tipo ZIP, imagens, etc.). O campo *binary* deverá ser preenchido com *true* caso o parâmetro seja binário ou *false* caso seja textual. O usuário deverá então utilizar o campo *textValue* para passar o valor textual ou o campo *binaryValue* para passar o valor binário. O campo *mimeType* é opcional, mas recomenda-se que seja preenchido com o tipo MIME do valor (exemplos: *application/zip, text/plain, text/xml,* etc.).

A resposta do método *submitProcess* traz o identificador do novo processo criado, que deverá ser usado posteriormente para se verificar seu *status* de execução e se obter seu resultado.

Para a verificação do *status* de execução de um processo, o programa do usuário deverá chamar o método *checkProcessResult* do serviço de *front-end* da plataforma BOWS. Esse método recebe como único parâmetro o identificador do processo e retorna seu *status*.

O último método disponível no serviço de *front-end* de BOWS é chamado *getProcessResult,* sendo responsável por retornar o resultado da execução de um processo. Esse método também recebe como parâmetro apenas o identificador do processo. O resultado é um objeto do tipo *ProcessResultResponse* que contém, além do resultado em si, outras informações relativas a ele. A Tabela 6 exibe os campos do objeto de resposta e suas descrições.

Tabela 6 - Campos do objeto *ProcessResultResponse*

Nome do campo	Descrição
processId	O identificador do processo.
applicationCode	O código da ferramenta que executou o processo.
resultText	O resultado textual da execução do processo (só preenchido se *isBinary = false).*
resultBinary	O resultado binário da execução do processo (só preenchido se *isBinary = true).*
isBinary	Determina se o resultado do processo é binário ou textual.
status	O *status* atual do processo.
message	Se houve erro na execução do processo, exibe a mensagem de erro.

Nota-se que da mesma forma que os valores dos parâmetros de submissão de um processo podem ser textuais ou binários, o resultado também o pode. O valor do campo *isBinary* determina se o resultado é binário ou textual. Caso seja binário, seu valor estará no campo *resultBinary.* Caso contrário, estará em *resultText.* O campo *status* poderá nesse caso

ser FINISHED ou ERROR. Caso seja ERROR, o processo apresentará o campo de resultados vazio e o campo *message* apresentará a mensagem de erro.

6.4 Transações de *Back-end* e o agente *Arrow*

As ferramentas registradas na plataforma BOWS devem ser idealmente executadas em *clusters* computacionais de alto desempenho. Esses *clusters* devem executar periodicamente um agente computacional que denominamos *arrow*, responsável por conectar ao servidor de BOWS e executar as operações de *back-end* para uma ferramenta específica registrada no BOWS. É recomendável que cada ferramenta possua seu próprio agente *arrow.* Nota-se aqui que a comunicação é sempre feita do *cluster* computacional para o servidor de BOWS. Isso evita a abertura de portas no *cluster*, o que torna BOWS um sistema extremamente seguro.

O agente *arrow* deve ser desenvolvido em qualquer linguagem de programação deve ser executado periodicamente por meio de uma ferramenta como o *crontab.* O agente deverá basicamente executar a seguinte sequencia de passos:

1. Buscar o próximo processo na fila de execução do BOWS.
2. Se um processo está disponível, modificar seu *status* para RUNNING.
3. Executar o processo obtido.
4. Gravar o resultado da execução do processo no BOWS.
5. Modificar o *status* do processo para FINISHED.

O serviço de *back-end* de BOWS disponibiliza cinco métodos, como demonstrado na Tabela 7. O cliente para o *back-end* de BOWS deve ser gerado por meio de seu documento WSDL, disponível na instalação padrão pelo caminho `services/BOWSAdmin?wsdl`.

Tabela 7 - Métodos de *back-end* da plataforma BOWS

Método	Descrição
createApplication	Registra uma nova ferramenta na plataforma BOWS.
removeApplication	Remove uma ferramenta da plataforma BOWS.
nextQueuedProcess	Retorna o próximo processo na fila de execução.
changeProcessStatus	Modifica o status de um processo.
insertProcessResult	Insere o resultado de um processo.

Os dois primeiros métodos são administrativos, e devem ser chamados, respectivamente, para o registro e remoção de uma ferramenta na plataforma BOWS. A seção 6.6 entra em maiores detalhes relativos a esses métodos.

Os demais três métodos do *back-end e* de BOWS devem ser usados na rotina do *script arrow*. A busca de novos processos é realizada por meio do método *nextQueuedProcess, o* qual retorna o próximo processo que está na lista de espera para execução. Esse método *nextQueuedProcess* aceita dois parâmetros, os quais são descritos na Tabela 8.

Tabela 8 - Parâmetros do método *nextQueuedProcess*

Parâmetro	Descrição
applicationCode	O código da ferramenta de onde o processo deve ser lido.
setToRunning	Informa se o status deve ser imediatamente modificado para RUNNING.

O método *changeProcessStatus* tem o objetivo de alterar o *status* de um projeto. Esse método deve ser chamado sempre que houver alguma alteração no andamento de um processo. Os possíveis *status* para um projeto são:

- *QUEUED:* Processo aguardando na fila de execução.
- *RUNNING:* Processo em execução.
- *FINISHED:* Execução do processo finalizada (resultado disponível).
- *ERROR:* Acusa que um erro ocorreu na execução do projeto.

A Tabela 9 lista e descreve os parâmetros aceitos pelo método *changeProcessStatus*.

Tabela 9 - Parâmetros do método *changeProcessStatus*

Parâmetro	Descrição
applicationCode	O código da ferramenta associada ao processo.
processId	O identificador do processo a ser alterado.
status	O novo *status* do processo.
errorMessage	Uma mensagem de erro, caso o *status* do projeto seja ERROR.

Finalmente, o método *insertProcessResult* deve ser usado pelo *script arrow* para a inserção do resultado final da execução de um processo na plataforma BOWS. A Tabela 10 lista e descreve os parâmetros aceitos por esse método.

Parâmetro	Descrição
applicationCode	O código da ferramenta associada ao processo.
processId	O identificador do processo que terá o resultado inserido.
resultText	O resultado da execução do processo em formato textual (deve ser preenchido somente se *isBinary* é falso)
resultBinary	O resultado da execução do processo em formato binário (deve ser preenchido somente se *isBinary* é verdadeiro)
mimeType	O *mime type* do resultado.
isBinary	Deve ser *true* se o resultado está em formato binário.

Tabela 10 - Parâmetros do método *insertProcessResult*

Como é possível observar na Tabela 10, o *script arrow* deverá informar se o resultado do processo é textual ou binário. O parâmetro *isBinary* deve ser usado par fazer essa discriminação. Se esse parâmetro tem valor *true*, o parâmetro a ser preenchido com o resultado da execução é *resultBinary.* Caso contrário, deve-se usar o parâmetro *resultText.* O campo *mime type* deve ser usado para a indicação do tipo MIME do resultado (ex: *text/plain, application/zip).*

6.5 Autenticação e segurança

Na plataforma BOWS, a segurança é garantida pelo fato de que o *cluster* que executa as ferramentas acessa o servidor BOWS de forma unidirecional. Portanto, não é necessário requisitar aos administradores do *cluster* aberturas de portas nem outras configurações que o deixem vulnerável. Desse modo, tanto o *cluster* quanto os programas ficam totalmente protegidos de acessos indevidos.

Para aumentar a confiabilidade do sistema, o *Web service* de *back-end* poderá ser acessível pelo protocolo *https*. Esse protocolo garante a encriptação dos dados trafegados, além de permitir que os acessos ao *back-end* só sejam realizados após autenticação por nome de usuário e senha. Dessa forma, as ferramentas registradas na plataforma BOWS e os resultados de cada usuário ficam protegidos de acessos não autorizados.

6.6 Registro de ferramentas no BOWS

Uma ferramenta de Bioinformática é registrada no BOWS por meio do método do *Web service* de *back-end createApplication.* Deve-se primeiramente definir um código de identificação único para a ferramenta (*applicationCode)* que será utilizado pelos usuários nas chamadas ao *front-end* de BOWS.

A Tabela 11 lista e descreve os parâmetros do método *createApplication.*

Tabela 11 - Parâmetros do método *createApplication*

Método	Descrição
code	O código identificador da ferramenta.
name	O nome da ferramenta (amigável).
description	Uma breve descrição da ferramenta.
paramConfigList	Lista de configuração de parâmetros

A lista de configuração de parâmetros deve receber um mais objetos do tipo *paramConfig.* Um objeto *paramConfig* configura um parâmetro que a ferramenta pode receber, além de possíveis restrições sobre seu valor. Por exemplo, um parâmetro chamado *fasta* pode ser configurado para receber apenas caracteres, em um formato compatível com o formato *fasta*.

A Tabela 12 exibe os campos de um objeto do tipo *paramConfig.*

Tabela 12 – Configuração de um parâmetro de processo

Parâmetro	Descrição
code	O código identificador da ferramenta.
name	O nome do parâmetro.
description	Uma breve descrição do parâmetro (opcional).
examples	Alguns exemplos de valores possíveis (opcional).
matcher	Uma expressão regular utilizada para validar o valor do parâmetro.
defaultValue	Um valor padrão para o parâmetro.
valueBinary	Define se o parâmetro deve possuir um valor binário.
mandatory	Define se este é um parâmetro obrigatório.

Por meio do *back-end* da plataforma BOWS também é possível remover uma aplicação existente. Para isso, deve-se chamar o método *removeApplication* com um único parâmetro: o código identificador da ferramenta.

6.7 Criação dos clientes de *Web service*

Os clientes para os serviços de *front-end* e *back-end* da plataforma BOWS podem ser facilmente criados em praticamente qualquer linguagem de programação moderna. Geralmente para toda linguagem há ferramentas disponíveis capazes de ler documentos do tipo WSDL e criar as classes necessárias para se efetuar chamadas ao serviço. Para a linguagem Java há por exemplo a biblioteca Apache CXF [47] que é capaz de ler documentos WSDL e criar as classes necessárias para o consumo do serviço. Na linguagem Perl, a biblioteca SOAP-WSDL [48] possui essa mesma função. Depois de criadas as classes para acesso aos serviços, o usuário poderá fazer chamadas aos métodos remotos de forma transparente, isso é, como se estivesse trabalhando com classes locais em seu programa.

A instalação padrão da plataforma BOWS disponibiliza os WSDLs nos seguintes endereços:

Front-end: `http://<servidor>:8080/services/BOWS?wsdl`

Back-end: `http://<servidor>:8080/services/BOWSAdmin?wsdl`

6.8 Instalação de BOWS

BOWS deve ser instalado em um servidor Web acessível a ambas as pontas. A plataforma é distribuída em um arquivo no formato WAR que pode ser instalado em qualquer servidor de aplicações Java (neste trabalho, foi utilizado o *container* Tomcat). Juntamente com a instalação do arquivo WAR, deve ser criado o banco de dados de BOWS. O *script* na linguagem SQL utilizado para a criação do banco é distribuído juntamente com a plataforma.

Os arquivos para a instalação da plataforma BOWS e as instruções de instalação podem ser encontradas no endereço https://sourceforge.net/projects/bows.

6.9 Estudo de caso

Para efeito de estudo de caso da plataforma BOWS, foi registrada na plataforma uma ferramenta para alinhamento múltiplo de sequências *Prank* [49]. A instalação da ferramenta na plataforma BOWS foi realizada por Ricardo Vialle do Laboratório de Biodados da UFMG.

A plataforma BOWS foi instalada e está disponível no servidor http://maxixe.icb.ufmg.br:8080. O endereço para acesso do documento WSDL de *front-end* é http://maxixe.icb.ufmg.br:8080/services/BOWS?wsdl.

O cliente de *back-end* (*script arrow)* foi construído por Vialle na linguagem *Matlab* e executa em outro servidor do laboratório. Ele também desenvolveu clientes de exemplo para a ferramenta *Prank* no BOWS. Foram construídos dois clientes de *front-end*, um na linguagem *Matlab* e outro na linguagem *Java*. Ambos os códigos estão também disponíveis no Apêndice deste trabalho. O cliente recebe como parâmetro um arquivo no formato MULTIFASTA, e submete um processo na plataforma BOWS para a ferramenta com o código "PlankAlign".

O binário do cliente *Java* (*.jar*) está disponível no endereço https://sourceforge.net/projects/bows. Para executá-lo em linha de comando deve-se utilizar a seguinte linha:

```
java -jar PrankBowsFrontend.jar arquivo.fasta
```

onde *arquivo.fasta* é um arquivo no formato Fasta.

Como exemplo, foi executado o cliente Java utilizando-se como entrada um arquivo MULTIFASTA contendo sequências de mioglobina de sete organismos diferentes. O conteúdo do arquivo de entrada é demonstrado na Figura 54, juntamente com a saída.

```
>Thunnus_albacares
MADFDAVLKCWGPVEADYTTMGGLVLTRLFKEHPETQKLFPKFAGIAQADIAGNAAISAH
GATVLKKLGELLKAKGSHAAILKPLANSHATKHKIPINNFKLISEVLVKVMHEKAGLDAG
GQTALRNVMGIIIADLEANYKELGFSG
>Euthynnus_pelamis
MADLDAVLKCWGAVEADFNTVGGLVLARLFKDHPETQKLFPKFAGITGDIAGNAAVAAHG
ATVLKKLGELLKAKGNHAAIIKPLANSHAKQHKIPINNFKLITEALAHVLHEKAGLDAAG
QTALRNVMGIVIADLEANYKELGFTG
>Scomber_japonicus
MADFDAVLKFWGPVEADYDKIGNMVLTRLFTEHPDTQKLFPKFAGIGLGDMAGNAAISAH
GATVLKKLAEVLKAKGNHAGIIKPLANSHATKHKIAINNFKLITEIIVKVMQEKAGLDAG
GQTALRNVMGVFIADMDANYKELGFSG
>Sarda_chiliensis
MADFDAVLKFWGPVEADYTSHGGLVLTRLFKEHPETQKLFPKFTGIAQADMAGNAAISAH
GATVLKKLGELLKAKGNHAAILKPMANSHATKHKIPINNFKLISEIIVKVMQEKAGMDAG
GQQALRNVMAAVIADLEANYKELGFSG
>Makaira_nigricans
MADFEMVLKHWGPVEADYATHGNLVLTRLFTEHPETQKLFPKFAGIAKADMAGNAAISAH
GATVLKKLGELLKAKGSHAAIIKPMANSHATKHKIPIKNFELISEVIGKVMHEKAGLDAA
GQKALKNVMTTIIADIEANYKELGFTG
>Bos_taurus
MGLSDGEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASE
DLKKHGNTVLTALGGILKKKGHHEAEVKHLAESHANKHKIPVKYLEFISDAIIHVLHAKH
PSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG
>Rattus_norvegicus
MGLSDGEWQMVLNIWGKVEGDLAGHGQEVLISLFKAHPETLEKFDKFKNLKSEEEMKSSE
DLKKHGCTVLTALGTILKKKGQHAAEIQPLAQSHATKHKIPVKYLEFISEVIIQVLKKRY
SGDFGADAQGAMSKALELFRNDIAAKYKELGFQG
```

```
>Thunnus_albacares
M----ADFDAVLKCWGPVEADYTTMGGLVLTRLFKEHPETQKLFPKFAGI-AQADIAGNA
AISAHGATVLKKLGELLKAKGSHAAILKPLANSHATKHKIPINNFKLISEVLVKVMHEK-
-AGLDAGGQTALRNVMGIIIADLEANYKELGFSG
>Euthynnus_pelamis
M----ADLDAVLKCWGAVEADFNTVGGLVLARLFKDHPETQKLFPKFAGI-T-GDIAGNA
AVAAHGATVLKKLGELLKAKGNHAAIIKPLANSHAKQHKIPINNFKLITEALAHVLHEK-
-AGLDAAGQTALRNVMGIVIADLEANYKELGFTG
>Scomber_japonicus
M----ADFDAVLKFWGPVEADYDKIGNMVLTRLFTEHPDTQKLFPKFAGI-GLGDMAGNA
AISAHGATVLKKLAEVLKAKGNHAGIIKPLANSHATKHKIAINNFKLITEIIVKVMQEK-
-AGLDAGGQTALRNVMGVFIADMDANYKELGFSG
>Sarda_chiliensis
M----ADFDAVLKFWGPVEADYTSHGGLVLTRLFKEHPETQKLFPKFTGI-AQADMAGNA
AISAHGATVLKKLGELLKAKGNHAAILKPMANSHATKHKIPINNFKLISEIIVKVMQEK-
-AGMDAGGQQALRNVMAAVIADLEANYKELGFSG
>Makaira_nigricans
M----ADFEMVLKHWGPVEADYATHGNLVLTRLFTEHPETQKLFPKFAGI-AKADMAGNA
AISAHGATVLKKLGELLKAKGSHAAIIKPMANSHATKHKIPIKNFELISEVIGKVMHEK-
-AGLDAAGQKALKNVMTTIIADIEANYKELGFTG
>Bos_taurus
MGLSDGEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASE
DLKKHGNTVLTALGGILKKKGHHEAEVKHLAESHANKHKIPVKYLEFISDAIIHVLHAKH
PSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG
>Rattus_norvegicus
MGLSDGEWQMVLNIWGKVEGDLAGHGQEVLISLFKAHPETLEKFDKFKNLKSEEEMKSSE
DLKKHGCTVLTALGTILKKKGQHAAEIQPLAQSHATKHKIPVKYLEFISEVIIQVLKKRY
SGDFGADAQGAMSKALELFRNDIAAKYKELGFQG
```

Figura 54 - Entrada e saída da ferramenta Prank na plataforma BOWS

O programa foi executado utilizando-se a seguinte linha de comando:

```
java -jar PrankBowsFrontend.jar mioglobinas.fasta
```

O resultado mostra o alinhamento múltiplo efetuado por meio da plataforma BOWS. Na região das inserções ou deleções foram incluídos os traços. É importante notar que tudo o que é requerido na máquina cliente é a instalação da máquina virtual *Java*. O usuário desse cliente não tem conhecimento de qual servidor na *Web* executou o alinhamento múltiplo. A execução pode, portanto, ser comandada por programas que disparem essa linha de comando.

Um segundo estudo de caso foi realizado na plataforma BOWS por meio da instalação de uma ferramenta de acesso ao programa *SeedServer*. Esse sistema, desenvolvido pelo pesquisador Rafael Lucas Muniz Guedes em seu trabalho de doutorado no Laboratório de Biodados da UFMG, tem como objetivo a criação de grupos de ortólogos a partir de uma sequência utilizada como semente. A processo e instalação do *SeedServer* é árduo pois o *software* contém muitas dependências. Por meio de BOWS, essa etapa é eliminada. Além disso, o processamento do *SeedServer* pode levar até duas horas para gerar um resultado. Por isso, para essa ferramenta foram desenvolvidos dois clientes: o primeiro submete o processo e o segundo obtém o resultado. Os dois clientes e o *script arrow* foram desenvolvidos na linguagem *Java* pelo pesquisador Ricardo Vialle.

O primeiro cliente aceita como entrada um ou mais identificadores de sequência *Uniprot*, separados por vírgula. O programa foi executado com a seguinte linha de comando:

```
java -jar SeedServerSubmitJob.jar Q9H9S0
```

O resultado da execução é número do processo criado:

```
PID = 127
```

De posse do número do processo, o próximo passo é chamar o segundo cliente, da seguinte forma:

```
java -jar SeedServerGetResults.jar 127
```

Caso o processo ainda não tenha sido executado, o programa exibe o *status* atual do mesmo, como abaixo:

```
RUNNING
```

Quando o resultado está disponível, o programa o exibe na tela, como no exemplo abaixo (exibidas somente as primeiras linhas do resultado):

```
Q01860   1         2         2         360      9606     1         1         1
PO5F1_HUMAN POU domain, class 5, transcription factor 1 OS=Homo sapiens
GN=POU5F1 PE=1 SV=1
O97552   1         0         1         360      9913     1         1         1
PO5F1_BOVIN POU domain, class 5, transcription factor 1 OS=Bos taurus
GN=POU5F1 PE=2 SV=1
P20263   1         0         1         352      10090    1         1         1
PO5F1_MOUSE POU domain, class 5, transcription factor 1 OS=Mus musculus
GN=Pou5f1 PE=1 SV=1
Q5TM49   1         0         1         360      9544     1         1         1
PO5F1_MACMU POU domain, class 5, transcription factor 1 OS=Macaca mulatta
GN=POU5F1 PE=3 SV=1
Q6MG27   1         0         1         352      10116    1         0         1
Q6MG27_RAT POU class 5 homeobox 1 OS=Rattus norvegicus GN=Pou5f1 PE=2 SV=1
```

6.10 Discussão

Neste capítulo, foi visto que a plataforma BOWS provê um vasto número de benefícios tanto para usuários quanto para criadores de ferramentas de Bioinformática. Os *Web services* são genéricos e utilizam uma interface comum para acesso a qualquer uma das ferramentas registradas. Desse modo, os usuários necessitam aprender apenas uma vez e estarão aptos a utilizar qualquer ferramenta. Além disso, os usuários não precisam lidar com os processos de instalação das ferramentas, que por vezes possuem muitas dependências, (como no caso do *software SeedServer*, descrito nos estudos de caso).

Devido à natureza universal dos *Web services*, as ferramentas registradas na plataforma BOWS podem ser acessadas praticamente de qualquer linguagem de programação moderna, uma vez que a maioria delas provê suporte para a construção de clientes de *Web services* a partir de documentos WSDL.

BOWS também protege o código e a propriedade intelectual das ferramentas pois o usuário final não tem acesso ao código e aos binários. Ademais, a arquitetura de BOWS impossibilita invasões aos *clusters* onde as ferramentas estão instaladas, já que as leituras são unidirecionais – sempre no sentido do servidor onde BOWS está instalado. Desse modo, a máquina onde BOWS está executando blinda os *clusters*.

Finalmente, o *back-end* de BOWS pode ser hospedado em super computadores, permitindo que biólogos e bioinformatas executem aplicações que demandam alto processamento diretamente de suas máquinas, o que de outro modo poderia ser inviável.

7. CONCLUSÕES

Neste trabalho foram desenvolvidas ferramentas computacionais com o objetivo de se inferir as origens de genes e vias metabólicas. Para se atingir esse objetivo foram criados algoritmos de LCA (mais próximo ancestral comum) que utilizam bancos de dados de genes ortólogos e de genomas completos.

Foi desenvolvido primeiramente o algoritmo de LCA simples, que busca o ancestral comum de um grupo de ortólogos. No entanto, as transferências horizontais se tornaram um obstáculo já que desviam o LCA obtido em direção à raiz da árvore da vida. Por isso, foi desenvolvido na sequencia o algoritmo de LCA múltiplo, capaz de lidar com esses eventos com a utilização de bases de genomas completos.

O trabalho se mostrou árduo uma vez que as bases de genes ortólogos disponíveis ainda se mostram bastante incompletas, assim como as bases de genomas completos. No entanto, essa é uma realidade que tende a se modificar nos próximos anos, à medida que aumente o número de sequenciamentos genômicos, os quais vêm se tornando cada vez mais rápidos e baratos.

Mesmo com a escassez de dados disponíveis, as ferramentas de LCA mostraram ótimos resultados em muitos casos e auxiliaram pesquisadores do Laboratório de Biodados da UFMG em suas pesquisas. Desse modo, espera-se que com a avalanche de novos dados que estão por vir, os resultados sejam cada vez melhores.

Além das ferramentas de análise de ancestralidade, foi também construída nesse trabalho a plataforma de *Web services* BOWS, capaz de centralizar e facilitar o acesso a ferramentas de bioinformática instaladas em *clusters* computacionais remotos. Essa plataforma já vem sendo usada com sucesso pelos pesquisadores do Laboratório de Biodados da UFMG, que podem assim disponibilizar suas ferramentas para a comunidade.

Por tudo isso, conclui-se que os objetivos deste trabalho foram atingidos com sucesso.

8. BIBLIOGRAFIA

1. GILBERT, W. Origin of life: The RNA world. **Nature,** v. 319, n. 6055, p. 618 (1986), 02 1986.

2. PIZZARELLO, S.; SHOCK, E. The organic composition of carbonaceous meteorites: the evolutionary story ahead of biochemistry. **Cold Spring Harb Perspect Biol,** v. 2, n. 3, p. a002105, Mar 2010. ISSN 1943-0264 (Electronic). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/20300213 >.

3. KAESSMANN, H. Origins, evolution, and phenotypic impact of new genes. **Genome Res,** v. 20, n. 10, p. 1313-26, Oct 2010. ISSN 1549-5469 (Electronic). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/20651121 >.

4. OHNO, S. **Evolution by Gene Duplication**. Springer-Verlag, 1970.

5. JACOB, F. Evolution and tinkering. **Science,** v. 196, n. 4295, p. 1161-6, Jun 10 1977. ISSN 0036-8075 (Print). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/860134 >.

6. LONG, M. et al. The origin of new genes: glimpses from the young and old. **Nat Rev Genet,** v. 4, n. 11, p. 865-75, Nov 2003. ISSN 1471-0056 (Print). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/14634634 >.

7. SIEPEL, A. Darwinian alchemy: Human genes from noncoding DNA. **Genome Res,** v. 19, n. 10, p. 1693-5, Oct 2009. ISSN 1549-5469 (Electronic). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/19797681 >.

8. ZHOU, Q. et al. On the origin of new genes in Drosophila. **Genome Res,** v. 18, n. 9, p. 1446-55, Sep 2008. ISSN 1088-9051 (Print). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/18550802 >.

9. TOLL-RIERA, M. et al. Origin of primate orphan genes: a comparative genomics approach. **Mol Biol Evol,** v. 26, n. 3, p. 603-12, Mar 2009. ISSN 1537-1719 (Electronic). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/19064677 >.

10. KNOWLES, D. G.; MCLYSAGHT, A. Recent de novo origin of human protein-coding genes. **Genome Res,** v. 19, n. 10, p. 1752-9, Oct 2009. ISSN 1549-5469 (Electronic). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/19726446 >.

11. CUNCHILLOS, C.; LECOINTRE, G. Integrating the universal metabolism into a phylogenetic analysis. **Mol Biol Evol,** v. 22, n. 1, p. 1-11, Jan 2005. ISSN 0737-4038 (Print). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/15356277 >.

12. SMITH, A. B.; PETERSON, K. J. Dating the time of origin of major clades: molecular clocks and the fossil record. **Annual Review of Earth and Planetary Sciences,** v. 30, n. 1, p. 65-88, 2002. ISSN 0084-6597.

13. PETERSON, K. J.; BUTTERFIELD, N. J. Origin of the Eumetazoa: testing ecological predictions of molecular clocks against the Proterozoic fossil record. **Proceedings of the National Academy of Sciences of the United States of America,** v. 102, n. 27, p. 9547-9552, 2005. ISSN 0027-8424.

14. DONNARD, E. et al. Preimplantation development regulatory pathway construction through a text-mining approach. **BMC Genomics,** v. 12, n. Suppl 4, p. S3, 2011. ISSN 1471-2164.

15. KANEHISA, M.; GOTO, S. KEGG: kyoto encyclopedia of genes and genomes. **Nucleic Acids Res,** v. 28, n. 1, p. 27-30, Jan 1 2000. ISSN 0305-1048 (Print). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/10592173 >.

16. NCBI Taxonomy Homepage. 2011. Disponível em: < http://www.ncbi.nlm.nih.gov/Taxonomy/ >. Acesso em: 12/05/2014.

17. FERNANDES, G. R. et al. A procedure to recruit members to enlarge protein family databases--the building of UECOG (UniRef-Enriched COG Database) as a model. **Genet Mol Res,** v. 7, n. 3, p. 910-24, 2008. ISSN 1676-5680 (Electronic). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/18949709 >.

18. SUZEK, B. E. et al. UniRef: comprehensive and non-redundant UniProt reference clusters. **Bioinformatics,** v. 23, n. 10, p. 1282-8, May 15 2007. ISSN 1367-4811 (Electronic). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/17379688 >.

19. MAGRANE, M.; CONSORTIUM, U. UniProt Knowledgebase: a hub of integrated protein data. **Database (Oxford),** v. 2011, p. bar009, 2011. ISSN 1758-0463 (Electronic). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/21447597 >.

20. KÖNIG, D. **Groovy in action**. Greenwich, Conn.: Manning**:** xxxiv, 659 p. p. 2007.

21. SMITH, G.; LEDBROOK, P. **Grails in action**. Greenwich, Conn.: Manning**:** xxix, 487 p. p. 2009.

22. AXMARK, D.; WIDENIUS, M.; MYSQL AB. **MySQL reference manual**. S.l.: O'Reilly**:** 814 p. p. 2002.

23. SEELY, S. **SOAP : cross platform Web service development using XML**. Upper Saddle River, NJ: Prentice Hall PTR, 2002. xiv, 391 p. ISBN 0130907634.

24. WALSH, A. E. **UDDI, SOAP and WSDL : the Web services specification reference book**. Upper Saddle River, NJ: Prentice Hall PTR, 2002. xx, 305 p. ISBN 0130857262.

25. RICHARDSON, L.; RUBY, S. **RESTful web services**. Farnham: O'Reilly, 2007. xxiv, 419 p. ISBN 9780596529260 (pbk.).

26. SAYERS, E. W. et al. Database resources of the National Center for Biotechnology Information. **Nucleic Acids Res,** v. 37, n. Database issue, p. D5-15, Jan 2009. ISSN 1362-4962.

27. WALLS, C.; BREIDENBACH, R. **Spring in action**. Greenwich, Conn.: Manning**:** xxviii, 444 p. p. 2005.

28. ALUR, D.; CRUPI, J.; MALKS, D. **Core J2EE patterns best practices and design strategies**. Java 2 platform, enterprise edition series. Upper Saddle River, NJ: Prentice Hall PTR**:** xxvi, 459 p. p. 2001.

29. DefaultMutableTreeNode. 2003. Disponível em: < http://docs.oracle.com/javase/1.4.2/docs/api/javax/swing/tree/DefaultMutableTreeNode.html >. Acesso em: 12/05/2014.

30. BALANI, N.; HATHI, R. **Apache CXF web service development develop and deploy SOAP and RESTful web services**. From technologies to solutions. Birmingham, U.K.: Packt Pub.**:** 1 online resource (vi, 319 p.) p. 2009.

31. KEGG. KEGG API. 2011. Disponível em: < http://www.genome.jp/kegg/soap/ >. Acesso em: 12/05/2014.

32. W3C. XML Path Language (XPath). 1999. Disponível em: < http://www.w3.org/TR/xpath/ >. Acesso em: 12/05/2014.

33. Bootstrap. 2014. Disponível em: < http://getbootstrap.com/ >. Acesso em: 12/05/2014.

34. jQuery. 2014. Disponível em: < http://jquery.com/ >. Acesso em: 12/05/2014.

35. BOSTOCK, M.; HEER, J. Protovis: A graphical toolkit for visualization. **Visualization and Computer Graphics, IEEE Transactions on,** v. 15, n. 6, p. 1121-1128, 2009. ISSN 1077-2626.

36. CXF plug-in for Grails. 2014. Disponível em: < http://grails.org/plugin/cxf >. Acesso em: 29/06/2014.

37. Homepage do Lab. Biodados UFMG 2011. Disponível em: < http://biodados.icb.ufmg.br >. Acesso em: 12/05/2014.

38. Web Services @ W3C. 2007. Disponível em: < http://www.w3.org/2002/ws/ >. Acesso em: 12/05/2014.

39. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). 2007. Disponível em: < http://www.w3.org/TR/2007/REC-soap12-part1-20070427/ >. Acesso em: 12/05/2014.

40. ASHBURNER, M. et al. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. **Nat Genet,** v. 25, n. 1, p. 25-9, May 2000. ISSN 1061-4036 (Print). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/10802651 >.

41. BARRELL, D. et al. The GOA database in 2009--an integrated Gene Ontology Annotation resource. **Nucleic Acids Res,** v. 37, n. Database issue, p. D396-403, Jan 2009. ISSN 1362-4962 (Electronic). Disponível em: < http://www.ncbi.nlm.nih.gov/pubmed/18957448 >.

42. FELSENSTEIN, J. et al. **The Newick tree format** 1986.

43. MADDISON, D. R.; SWOFFORD, D. L.; MADDISON, W. P. NEXUS: an extensible file format for systematic information. **Systematic Biology,** v. 46, n. 4, p. 590-621, 1997. ISSN 1063-5157.

44. ZMASEK, C. M.; CANNON, E. K. S. phyloXML| Describing Annotated Phylogenetic Trees With XML.

45. ECMA-404. **The JSON Data Interchange Format:** 8 p. 2013.

46. DEHAL, P. S. et al. MicrobesOnline: an integrated portal for comparative and functional genomics. **Nucleic acids research,** v. 38, n. suppl 1, p. D396-D400, 2010. ISSN 0305-1048.

47. Apache CXF. 2014. Disponível em: < http://cxf.apache.org/docs/wsdl-to-java.html >. Acesso em: 06/06/2014.

48. SOAP-WSDL-2.00.10. 2014. Disponível em: < http://search.cpan.org/~mkutter/SOAP-WSDL-2.00.10/ >. Acesso em: 06/06/2014.

49. LÖYTYNOJA, A.; GOLDMAN, N. webPRANK: a phylogeny-aware multiple sequence aligner with interactive alignment browser. **Bmc Bioinformatics,** v. 11, n. 1, p. 579, 2010. ISSN 1471-2105.

Apêndice A – TRECHOS DE CÓDIGO

Back-end da ferramenta Prank Align

O código abaixo constitui o *script Arrow* da ferramenta Prank Align, implementado em *Matlab* pelo doutorando Ricardo Vialle, do Labotatório de Biodados da UFMG.

```
function prank_bows_backend(prankpath)
import @BOWSAdminService.*
try
    % Get next job to run
    torun = nextQueuedProcess(BOWSAdminService,'PRANKALIGN','1');
    % Run PRANK
    fastafilename = [torun.applicationCode '_' torun.processId '.fasta'];
    outputfile =  [torun.applicationCode '_' torun.processId '_output'];
    fastaencoded = torun.params.textValue;
    fasta = char(base64decode(fastaencoded));
    fid = fopen(fastafilename,'w');
    fwrite(fid,fasta);
    fclose(fid);
    prank(fastafilename,prankpath,outputfile);
    outputfileprank =  [outputfile '.best.fas'];
    % Prepare result
    processResultParam.processId = torun.processId;
    processResultParam.applicationCode = torun.applicationCode;
    processResultParam.resultText =  base64file(outputfileprank);
    processResultParam.resultBinary = '';
    processResultParam.mimeType = '';
    processResultParam.isBinary = '0';
    % Send result
    insertProcessResult(BOWSAdminService,processResultParam);
    % Remove files
    delete(outputfileprank);
    delete(fastafilename);
    % Change status
    changeProcessStatus(BOWSAdminService,torun.processId,torun.applicationCode,
                        'FINISHED','Error');
catch e
    disp(e)
end
```

***Front-end* da ferramenta Prank Align em Matlab**

O código abaixo constitui o cliente da ferramenta Prank Align, implementado em *Matlab* pelo doutorando Ricardo Vialle, do Labotatório de Biodados da UFMG.

```
function prank_bows_frontend(filename)
import @BOWSService.*
try
    % Prepare variables
    submitProcessParam.applicationCode = 'PRANKALIGN';
    submitProcessParam.parameter(1).name = 'Fasta';
    submitProcessParam.parameter(1).textValue = base64file(filename);
    submitProcessParam.parameter(1).binaryValue = '';
    submitProcessParam.parameter(1).mimeType = '';
    submitProcessParam.parameter(1).binary = '0';
    % Submit job
    pid = submitProcess(BOWSService,submitProcessParam.applicationCode,
                        submitProcessParam.parameter);
    % Loop checking status until FINISHED
    while 1
        processStatus = checkProcessStatus(BOWSService,pid.processId);
        if strcmpi(processStatus.status,'FINISHED')
            break;
        end
        pause(5);
    end
    % Retrieve result
    processResult = getProcessResult(BOWSService,pid.processId);
    % Decode result
    result = char(base64decode(processResult.resultText));
    % Write on file
    fid = fopen([filename '.result'],'w');
    fwrite(fid,result);
    fclose(fid);
end
```

Front-end da ferramenta Prank Align em Java

O código abaixo constitui o cliente da ferramenta Prank Align, implementado em *Java* pelo doutorando Ricardo Vialle, do Labotatório de Biodados da UFMG.

```
1  import java.io.*;
2  import bows.*;
3  import org.apache.commons.codec.binary.Base64;
4
5  public class PrankBowsFrontend {
6
7      public static void main(String[] args) throws Exception {
8          String applicationCode = "PRANKALIGN";
9          // Convert fasta file to base64 code
10         String text = encodeFileToBase64Binary(filename);
11         // Declare webservice object
12         BOWSServicePortTypeProxy service = new BOWSServicePortTypeProxy();
13         // Parameters object
14         SubmissionParam submitProcessParam = new SubmissionParam();
15         submitProcessParam.setMimeType("");
16         submitProcessParam.setName("Fasta");
17         submitProcessParam.setTextValue(text);
18         SubmissionParam[] submitProcessParamArray  = new SubmissionParam[1];
19         submitProcessParamArray[0] = submitProcessParam;
20         // Submit job
21         SubmissionResponse submitResponse = new SubmissionResponse();
22         submitResponse = service.submitProcess(applicationCode,submitProcessParamArray);
23         // Loop checking status until FINISHED
24         long processId = submitResponse.getProcessId();
25         while(true){
26             if (new String("FINISHED").equals(service.
27                                         checkProcessStatus(processId).getStatus().toString())){
28                 break;
29             }
30             Thread.sleep(5000);
31         }
32         // Retrieve result
33         ProcessResultResponse resultResponse = new ProcessResultResponse();
34         resultResponse = service.getProcessResult(processId);
35         // Decode result and write on file
36         byte dearr[] = Base64.decodeBase64(resultResponse.getResultText());
37         FileOutputStream fos = new FileOutputStream(new File(filename+".result"));
38         fos.write(dearr);
39         fos.close();
40     }
41
42 }
```

Apêndice B – TRABALHO SUBMETIDO À REVISTA BIOINFORMATICS

Application Notes

BOWS to centralize Bioinformatics tools in Web services

Henrique Velloso*, Ricardo Vialle and J. Miguel Ortega

Departmento de Bioquímica e Imunologia, Instituto de Ciências Biológias, UFMG, Av. Antônio Carlos 6627 – Belo Horizonte, MG - Brazil

ABSTRACT

High Performance Computing (HPC) has become available in many centers in Latin America. Especially Brazil hosts a National HPC System (SINAPAD), which is available as an open resource for the local scientific community, organized in AB3C, as well HPC clusters are accessible to the Iberoamerican community associated in SoI-Bio. Here we describe a generic system based on Web Services named BOWS (Bioinformatics Open Web Services), which allows access to public and authorial applications running on HPC clusters. BOWS runs in a front-end server (named bow) and provides asynchronous web service requisitions. BOWS allows a quick incorporation of several independent applications since programmers can install them in HPC clusters in any programming language. The lonely requirement is to write a script named "arrow" which calls BOWS backend services periodically in order to check for new processes and the required parameters. If a new process is found, the "arrow" script should change the requisition status from waiting to running, run the process in the HPC cluster and, when the job is complete, call a BOWS backend service to populate write back the results. The results will then be available to the requestor. On the other hand BOWS generates a generic WSDL descriptor file, which allows language-independent submissions for any registered application. The installation files for the BOWS platform and installation instructions are available at https://sourceforge.net/projetcs/bows.

1 INTRODUCTION

Bioinformatics is the study of informatics contained in living being. Dealing with an increasing amount of data provided by massive DNA sequencing and with the novelty of patterns exposed by systemic approaches demands using powerful tools. On the other side, there are biologists and bioinformaticians who would be largely benefited with these programs for they researches, but usually face a wide range of difficulties before having these tools running and producing results.

They usually have to deal with complex installation procedures as different tools often rely on specific system libraries. Also, bioinformaticians are forced to become familiar with different operating systems, since many tools are OS-specific. Other problem faced is that many tools demand a high processing unit to deal with elevated amounts of data, which often is not available. Moreover, each application has its own learning curve, forcing users to spend time learning how to use each one.

Bioinformatics centers are present all around the world, such as the KEGG Encyclopedia (Kanehisa and Goto, 2010), the Uniprot database (Uniprot, 2008) and the ExPASy (Gasteiger *et al.,*2003).

*To whom correspondence should be addressed

Bioinformaticians would largely benefit of a system that could centralize most of the tools needed for their works, using a unique easy interface for input and output. Web services, due to their universal nature and widely known interface, constitute a very good option to achieve this goal. Furthermore, Web services provide a security layer that can restrict access to tools and protect copyrighted programs running on their backend.

Here we present BOWS (Bioinformatics Open Web Services), a Web services platform that allows centralized and standardized access to Bioinformatics tools. BOWS is installed in an intermediary machine and provides a front-end to Bioinformatics tools' consumers and a secure back-end to tools' owners. The front-end exposes a WSDL with three Web methods used by Bioinformaticians to submit processes check status and get results from applications. The back-end consists of five secure Web methods used by tool owners to register new applications, delete existing applications, read new processes, change status and submit results. This model creates a cyclical process where submissions are sent to BOWS via front-end services and then asynchronously read and executed by the back-end server. When the results are ready they are written back to BOWS via back-end services, becoming available to the requester.

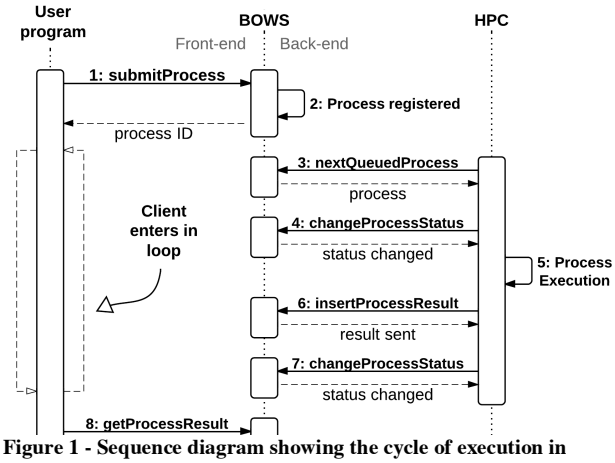


Figure 1 - Sequence diagram showing the cycle of execution in BOWS. Front-end transactions are in the left while back-end transactions are shown in the right.

2 BOWS TRANSACTIONS

BOWS server offers two different WSDL files: a front-end WSDL and a back-end WSDL. Front-end transactions are made by client programs to submit new processes to registered applications

and read results. Back-end transactions are performed by HPCs to retrieve submitted processes and insert results after execution.

2.1 Back-end transactions

Since intensive computing is often required by Bioinformatics applications, BOWS was designed to deal with asynchronous WebService requisitions. To add high performance computing to the system through a safe and simple way, requisitions are received by the BOWS front-end services and stored in a local database. An agent in the computational cluster named "arrow" is responsible for connecting with the back-end services. To allow back-end transactions, BOWS makes available a restricted back-end Web service with three Web methods. The "arrow" agent access the back-end Web service to looks for new jobs using the "readProcess" method. If one is found, the agent modifies the status from “waiting” to “running” with the "changeStatus" operation and processes the input in the computational cluster. When the result is available, it saves the result back to BOWS using the "setResult" Web method and finally changes the status to “finished”. If a processing error is found, the status can also be changed to "error" with the appropriate error message.

Safety is granted by the fact that the cluster connects unidirectionally to the BOWS server, thus it is not necessary to request to cluster administrators any vulnerable setting such as opening additional ports. Set up is simple since both the application and back-end connecting agent are written in any computational language. *Crontab* triggers the agent periodically (typically every minute). Moreover, the number of nodes typically required by the application is defined in the agent. Virtually the HPC user does not require any special management to make the application to be linked to BOWS.

The back-end services should also be used to register new applications to BOWS. There are two administrative methods created to allow application owners set up and/or remove a new application: *createApplication* and *removeApplication.* Thus, this step should be performed before executing processes.

2.2 Front-end transactions

BOWS front-end services provides Web methods that allow users submitting processes to registered applications and reading results. As the responses are asynchronous, typically the user program should enter a loop waiting for the availability of a process result. In general, the user program should operate as follows to access a tool in the BOWS platform:

1. Submit a new process with the desired parameters calling the front-end Web method *submitProcess.*
2. Enter a loop where at each iteration it should check the status of the submitted process calling the method *checkProcess.*
3. If the process status is FINISHED, it should call the method *getResults* to obtain the results.
4. If the process status is ERROR, it should handle the error.

3 AUTHENTICATION AND SECURITY

In BOWS platform, security is guaranteed by the fact that the cluster running the tools accesses the server BOWS unidirectional. Therefore, it is not necessary to request cluster administrators doorways or other settings that leave vulnerable. Thus, both the cluster as the programs are fully protected from unauthorized access.

To increase system reliability, the Web service backend may be accessible via https protocol. This protocol ensures the encryption of data traffic. Access to the back end will be realized only after authentication by username and password. Thus, the tools registered in BOWS platform and the results of each user are protected from unauthorized access.

4 EXAMPLE

A case study was conducted in the BOWS platform. The multiple sequence alignment Prank was registered. Ricardo Vialle performed the installation of the BOWS platform. Vialle also built the backend script (arrow) in Matlab language and which is executed on another server in the lab. He also developed sample clients to Prank tool in BOWS. Two clients for front end, one in Matlab language and the other in Java were built. The client receives as a parameter a file in MULTIFASTA format and submits a process to the BOWS platform with the code "PlankAlign". As an example, it has run the Java client using as input a file containing MULTIFASTA myoglobin sequences from seven different organisms.

The results show the multiple alignments performed by BOWS platform. In the region of insertions or deletions traces were included. It is important to note that all that is required on the client machine is the installation of the Java virtual machine. The user of this client is not aware of any web server executed the multiple alignment.

5 CONCLUSIONS

BOWS provides a wide range of benefits to both Bioinformatics tools consumers and owners. The Web methods are generic and use a common syntax to access any registered application, so users only have to learn once and are ready to use any tool. Also, due to the universal nature of Web services, BOWS' registered applications can be accessed from virtually any programming language, as most of them provide extensive support to build Web service clients. This solution also protects the code and intellectual property of applications, since the final user do not have access to the code or the binaries. Finally, the backend can run in a high performance computers, letting biologists and bioinformaticians remotely run high-processing demand applications directly from their machines, what otherwise would be unfeasible.

REFERENCES

Kanehisa, M. and Goto, S. (2000) KEGG: kyoto encyclopedia of genes and genomes, *Nucleic acids research*, **28**, 27-30.

UniProt, C. (2008) The universal protein resource (UniProt), *Nucleic acids research*, **36**, D190-D195.

Gasteiger, E*., et al.* (2003) ExPASy: the proteomics server for in-depth protein knowledge and analysis, *Nucleic acids research*, **31**, 3784-3788.