Ministério da Educação
Universidade Tecnológica Federal do Paraná
Pró-Reitoria de Pesquisa e Pós-Graduação

Relatório Final de Atividades

Integração de dados

Inferência de GRNs a partir da integração dos dados da Arabidopsis thaliana

Euler Angelo de Menezes Junior
Bolsista CNPq
Tecnologia em Analise de Desenvolvimento de Sistemas
Data de ingresso no programa: 08/2014
Prof. Dr. Fabrício Martins Lopes

Bioinformática

CAMPUS CORNÉLIO PROCÓPIO, 2015

Integração de dados

Euler Angelo de Menezes Junior, Fabrício Martins Lopes

Universidade Tecnológica Federal do Paraná – UTFPR
Campus Cornélio Procópio
Avenida Alberto Carazzai, 1640 – Centro, Cornélio Procópio – PR, 86300-000

euler@alunos.utfpr.edu.br, fabricio@utfpr.edu.br

Palavras-chave: Integração de dados, Arabidopsis thaliana, Bioinformática

INTRODUÇÃO

Na era da informação, a quantidade de dados produzida é tão grande que supera a quantidade que conseguimos processar e analisar. Gerenciar e processar dados que crescem exponencialmente em tamanho em um curto período de tempo [1], é um dos maiores desafios da informática.

Isso não é diferente no contexto de dados biológicos, onde a tecnologia de aquisição das expressões gênicas vêm evoluindo rapidamente, sendo possível obter a expressão gênica de milhares de genes simultaneamente e em múltiplos instantes de tempo.

Com a grande quantidade de dados gerados, eles foram espalhados por vários bancos de dados públicos, como TAIR, KEGG e NCBI, porém muitos em formatos diferentes, dificultando a aplicação algoritmos computacionais.

Sendo assim, viu-se a necessidade de integrar todos esses dados em um único banco de dados, categorizar de uma forma mais simplificada e indexar as informações com identificadores únicos e numéricos para uma possível aplicação de fórmulas matemáticas e estatísticas.

Na bioinformática, a inferência de redes de regulação gênica (GRNs) a partir de seus perfis de expressão é um problema a ser solucionado, sendo necessário o desenvolvimento de métodos alternativos para montar as redes de forma mais adequada e precisa.

Tomamos como estudo de caso a planta *Arabidopsis thaliana*, onde é feita a integração dos dados biológicos obtidos dos bancos de dados públicos, e disponibilização dos mesmos para visualização em uma interface WEB, tão como uma API onde a partir dela é possível montar e inferir as GRNs.

DESENVOLVIMENTO

Esse trabalho consiste em integrar os dados em um único banco, eliminando informações duplicadas e indexar de uma forma que seja possível descobrir os inter-relacionamentos entre os genes.

O projeto foi dividido entre as seguintes etapas:

1. Identificação das informações a serem extraídas
2. Extração dos dados
3. Montagem do banco de dados
4. Eliminação de informações duplicadas e indexação dos dados
5. Disponibilização dos dados para pesquisa

Depois da seleção das informações a serem extraídas para cada banco, foi feito o trabalho de extração dos dados do banco de dados TAIR e KEGG para cada gene (identificado pelo seu Locus), no caso de existir o NCBI GeneID de algum gene nas informações obtidas do KEGG, executou-se a extração dos dados daquele gene no NCBI. A tecnologia utilizada foi NodeJS [2], com os módulos request [3], para baixar o HTML, e cheerio [4], para extrair os dados navegando pelo DOM da página. As informações foram salvas no formato JSON [5], contendo o locus identificador e um vetor de informações para cada categoria de informação.

Para o banco de dados, foi utilizado um banco relacional, PostgreSQL [6], contendo uma tabela com todos os genes, uma tabela para cada categoria de informação e, por fim, várias tabelas de relacionamento entre os genes e cada categoria de informação. Em todas as tabelas foram colocados chaves primárias numéricas (identificadores). A escolha de um banco de dados relacional com identificadores numéricos se deu pela facilidade de execução de algoritmos sobre esses identificadores, podendo assim gerar saídas com somente números para cálculos e estatísticas. Foram também inseridas VIEWS para listar os genes e suas informações relacionadas.

Na inserção, foi executado um script, também em NodeJS, para ler os dados no formato JSON, verificar a existência de duplicados e inserir no banco criado em PostgreSQL.

Com o banco devidamente montado e populado, deu-se início ao desenvolvimento de uma interface WEB e uma API JSON, utilizando o framework Express [7] para NodeJS, juntamente com pg-query [8] para conexão com o banco de dados. Nelas são possíveis a listagem e visualização de todos os dados e utilização dos mesmos em outros sistemas e ambientes computacionais.

RESULTADOS E DISCUSSÃO

Na extração dos dados, foi observada a dificuldade de manter o software desenvolvido, pois, sempre que ocorrerem mudanças nas estruturas HTML dos bancos de dados, terá que ser revisado e muitas vezes corrigido o algoritmo de extração, uma solução é pegar os dados de algum outro formato, como XML, porém não existem muitas opções disponíveis em todos os bancos de dados.

Links utilizados para extração dos dados:

- TAIR: http://arabidopsis.org/servlets/TairObject?type=locus&name=LOCUS
- KEGG: http://www.genome.jp/dbget-bin/www_bget?ath:LOCUS
- NCBI: http://www.ncbi.nlm.nih.gov/gene/NCBIGENEID

De cada campo, foram selecionadas algumas informações, separadas por categorias:

- TAIR: description, gene_model_type, other_names, located_in, functions_in, has;
- KEGG: definition, module, other_dbs, pathway;
- NCBI: official_symbol, official_full_name, see_related, gene_symbol, gene_description, primary_source, locus_tag, gene_type, rna_name, refseq_status, organism, lineage, also_known_as, summary, sequence, location, id_protein.

No Apêndice A temos o código para extração dos dados, montado como um módulo para o NodeJS, contendo uma função para cada banco de dados onde, para o TAIR e KEGG, passamos o LOCUS e a função para tratamento dos dados extraídos e, para o NCBI, passamos o NCBI-GeneId e a função para tratamento dos dados. Os HTMLs baixados foram salvos caso necessário o uso posterior. Caso nenhuma função de tratamento definida, é executada uma função para salvar no formato JSON.

Os arquivos criados seguiram uma estrutura predefinida de subdiretórios, sendo arquivos/BANCO/html/LOCUS.html (ex.: arquivos/kegg/html/AT1G01010.html) e arquivos/BANCO/json/LOCUS.json (ex.: arquivos/tair/json/AT1G01010.json), no caso do NCBI, substituindo o locus pelo geneid.

Abaixo o código do TAIR, na linha 9 é definido 2 parâmetros, o locus e a função de callback, nela executamos o módulo request com o link para o banco de dados.

```
9.      TAIR: function(locus, callback) {
10.         request('http://arabidopsis.org/servlets/TairObject?type=locus&name=' + locus, function (error, response, html) {
11.             var obj = { db : 'tair',
12.                         locus : locus,
13.                         description : [],
14.                         gene_model_type : [],
15.                         other_names : [],
16.                         located_in : [],
17.                         functions_in : [],
18.                         has : []};
```

Como resultado obtemos o conteúdo HTML, junto com os códigos de resposta ou algum erro, para cada informação a ser extraída, foi inicializado um vetor vazio e salvo no objeto JSON.

```
19. if (typeof callback === 'undefined') {
20.                 callback = save;
21.             }
```

Caso nenhuma função callback seja passada, é utilizada uma predefinida (save).

```
22. if (!error && response.statusCode == 200) {
23.                 var $ = cheerio.load(html);
24.                 var split = {};
25.
26.                 split.description = '; '
27.                 split.other_names = ',  ';
28.                 split.annotations = ',  ';
29.
30.                 var tmp = '';
31.                 $('.clearfix').first().contents().find('tr').each(function(i, elem) {
32.                     var aux = $(this).find('th').text().trim().toLowerCase().split(' ').join('_').replace(':', '');
33.                     if(obj[aux]) {
34.                         obj[aux] = $(this).find('th').next().next().text().trim().split(split[aux]);
35.                     }
36.                     if(aux == 'annotations_category relationship_type_ keyword') {
37.                         $(this).find('tr').each(function(i, elem) {
38.                             var aux2 = $(this).find('td').first().next().next().text().trim().toLowerCase().split(' ').join('_');
39.                             if(obj[aux2]) {
```

```
                                var aux3 = $(this).find('td').first().next().next().
    next().next().text().trim().split(split.annotations);
                                obj[aux2] = obj[aux2].concat(aux3);
                            }
                        });
                    }
                });
                callback(null, obj);
            } else {
                callback(error, obj);
            }
```

Caso nenhum erro ocorrido, navegamos pelo DOM do HTML a fim de capturar as informações, onde cada informação encontrada é inserida no vetor definido anteriormente.

Por fim, chamamos a função callback passando o erro junto com um objeto vazio, ou nulo quando nenhum erro juntamente com o objeto populado.

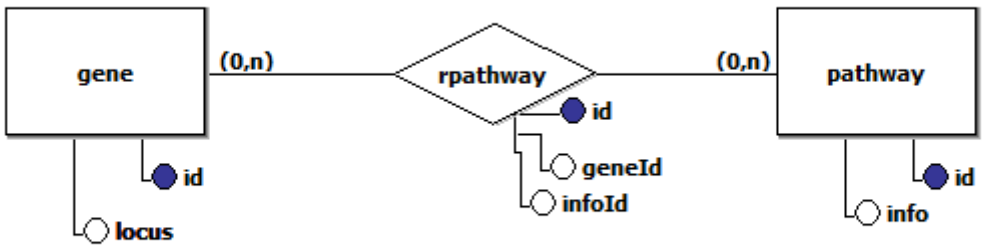
O arquivo HTML é salvo automaticamente pelo request utilizando seu método pipe.

```

        }).pipe(fq.createWriteStream('arquivos/html/tair/'+locus+'.html'));
```

No banco de dados, os resultados foram satisfatórios, apesar de não ter nenhum problema a ser corrigido, o ideal é a utilização de algum padrão de banco para dados biológicos, ou, melhor estruturação desse mesmo.

Nele foi criada uma tabela para todos os genes, contento uma chave primária numérica juntamente com o LOCUS, uma tabela para cada categoria de informação, também com uma chave primária mais a string da informação, e, para cada informação, uma tabela de relacionamento. Como exemplo, diagrama dos genes com as informações “pathway” extraídas do KEGG na imagem abaixo:



A existência de dados duplicados foi verificada antes da inserção, dada uma informação qualquer de um gene, foi dado em SELECT em sua respectiva tabela a procura dela, inserida caso inexistente, criando também o relacionamento com o gene e, caso encontrada, foi dado outro SELECT na tabela de relacionamentos, onde era inserido caso inexistente.

O script completo para criação das tabelas encontra-se no Apêndice B, e para as views no Apêndice C.

Os genes foram inseridos manualmente utilizando um script SQL com os códigos de inserção (INSERT INTO gene (locus) VALUES (‘AT1G01010’)), já o script para inserção das informações pode ser visualizado no Apêndice D.

No banco, não há diferenciação das informações para o TAIR, KEGG e NCBI além de cada categoria ter uma tabela e um relacionamento separado, nesse quesito, uma melhoria a ser feita, é identificar a duplicidade de informações entre tabelas diferentes, pois como a

estrutura dos dados foi mantida a mesma, pode haver informações duplicadas em formatos diferentes.

A interface WEB para visualização dos dados é satisfatória, podendo abrir espaço para mais funcionalidades como gerar GRNs.

A API funciona como o esperado, via web, podendo ser utilizada para gerar as tabelas, em qualquer linguagem que consiga ler objetos JSON, tornando os dados facilmente acessíveis.

Tanto a interface WEB como a API oferecem os seguintes formatos:

- Lista de genes
- Lista de informações por categoria
- Lista de informações compartilhadas entre genes
- Lista de informações por gene
- Lista de informações compartilhadas entre dois ou mais genes

Como resultado dos trabalhos, os identificadores das informações foram extraídos para uma tabela em texto, sendo utilizados no desenvolvimento de um artigo (Apêndice E), aceito no CIARP 2015.

AGRADECIMENTOS

Agradeço meu orientador Fabrício Martins Lopes, pelo apoio e orientação nas pesquisas, a Universidade Tecnológica Federal do Paraná (UTFPR) e ao órgão CNPq.

REFERÊNCIAS

[1] Ian Gorton, Paul Greenfield, Alex Szalay, Johns Hopkins. **Data-Intensive Computing in the 21st Century.** <http://www.computer.org/csdl/mags/co/2008/04/mco2008040030.html#bibmco20080400301>

[2] Node.js Foundation. **Node.JS** <https://nodejs.org/en/>

[3] Lalit Kapoor, Loïc Mahieu, Mikeal Rogers, Maciej Małecki, James Nylen, Sean Hagstrom, simo, eriksm. **Request** <https://github.com/request/request>

[4] Matt Mueller. **Cheerio** <https://github.com/cheeriojs/cheerio>

[5] Ecma International. **JSON** <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>

[6] The PostgreSQL Global Development Group. **PostgreSQL** <http://www.postgresql.org/>

[7] StrongLoop. **Express** <http://expressjs.com/>

[8] Brian Carlson. **Node pg-query** <https://github.com/brianc/node-pg-query>

Apêndice A – Mecanismo de busca e extração das informações

```
1.  var request = require('request');
2.  var cheerio = require('cheerio');
3.  var fs = require('fs');
4.  var FileQueue = require('filequeue');
5.  var fq = new FileQueue(100);
6.
7.  module.exports = {
8.
9.      TAIR: function(locus, callback) {
10.         request('http://arabidopsis.org/servlets/TairObject?type=locus&name=' + locus,
        function (error, response, html) {
11.             var obj = { db : 'tair',
12.                         locus : locus,
13.                         description : [],
14.                         gene_model_type : [],
15.                         other_names : [],
16.                         located_in : [],
17.                         functions_in : [],
18.                         has : []};
19.             if (typeof callback === 'undefined') {
20.                 callback = save;
21.             }
22.             if (!error && response.statusCode == 200) {
23.                 var $ = cheerio.load(html);
24.                 var split = {};
25.
26.                 split.description = '; '
27.                 split.other_names = ',  ';
28.                 split.annotations = ',  ';
29.
30.                 var tmp = '';
31.                 $('.clearfix').first().contents().find('tr').each(function(i, elem) {
32.                     var aux = $(this).find('th').text().trim().toLowerCase().split(' ')
    .join('_').replace(':', '');
33.                     if(obj[aux]) {
34.                         obj[aux] = $(this).find('th').next().next().text().trim().split
    (split[aux]);
35.                     }
36.                     if(aux == 'annotations_category relationship_type_ keyword') {
37.                         $(this).find('tr').each(function(i, elem) {
38.                             var aux2 = $(this).find('td').first().next().next().text().
    trim().toLowerCase().split(' ').join('_');
39.                             if(obj[aux2]) {
40.                                 var aux3 = $(this).find('td').first().next().next().nex
    t().next().text().trim().split(split.annotations);
41.                                 obj[aux2] = obj[aux2].concat(aux3);
42.                             }
43.                         });
44.                     }
45.                 });
46.                 callback(null, obj);
47.             } else {
48.                 callback(error, obj);
49.             }
50.
51.         }).pipe(fq.createWriteStream('arquivos/html/tair/'+locus+'.html'));
52.     },
53.
54.     KEGG: function(locus, callback) {
55.         request('http://www.genome.jp/dbget-
    bin/www_bget?ath:' + locus, function (error, response, html) {
56.             var obj = { db : 'kegg',
```

```
                        locus : locus,
                        definition : [],
                        module : [],
                        other_dbs : [],
                        pathway : []};
            if (typeof callback === 'undefined') {
                callback = save;
            }
            if (!error && response.statusCode == 200) {
                var $ = cheerio.load(html);
                var split = {};

                split.definition = ',';
                split.module = ' ';
                split.other_dbs = ' ';
                split.pathway = ' ';

                var tmp = '';
                $('.th10').each(function(i, elem) {
                    var aux = $(this).text().toLowerCase().trim().split(' ').join('_');

                    var aux2 = $(this).next();
                    if (obj[aux]) {
                        if(aux == 'other_dbs') {
                            aux2.find('div').each(function(i, elem) {
                                if(i % 2 != 0) {
                                    var odb = {
                                        name : $(this).text().trim().replace(':', ''),

                                        id : $(this).next().text().trim()
                                    }
                                    obj.other_dbs = obj.other_dbs.concat(odb);
                                }
                            });
                        } else if(aux == 'pathway' || aux == 'module') {
                            aux2.find('tr').each(function(i, elem) {
                                obj[aux] = obj[aux].concat($(this).text().trim().replac
e( /[\s\n\r]+/g, ' ' ));
                            });
                        } else {
                            aux2.find('div').each(function(i, elem) {
                                obj[aux] = obj[aux].concat($(this).text().trim());
                            });
                        }
                    }
                });
                $('.th11').each(function(i, elem) {
                    var aux = $(this).text().toLowerCase().trim().split(' ').join('_');

                    var aux2 = $(this).next();
                    if (obj[aux]) {
                        if(aux == 'other_dbs') {
                            aux2.find('div').each(function(i, elem) {
                                if(i % 2 != 0) {
                                    var odb = {
                                        name : $(this).text().trim().replace(':', ''),

                                        id : $(this).next().text().trim()
                                    }
                                    obj.other_dbs = obj.other_dbs.concat(odb);
                                }
                            });
                        } else if(aux == 'pathway' || aux == 'module') {
                            aux2.find('tr').each(function(i, elem) {
                                obj[aux] = obj[aux].concat($(this).text().trim().replac
e( /[\s\n\r]+/g, ' ' ));
```

```
                            });
                        } else {
                            aux2.find('div').each(function(i, elem) {
                                obj[aux] = obj[aux].concat($(this).text().trim());
                            });
                        }
                    }
                });

                callback(null, obj);
            } else {
                callback(error, obj);
            }
        }).pipe(fq.createWriteStream('arquivos/html/kegg/'+locus+'.html'));
    },

    NCBI: function(geneid, callback) {
        request('http://www.ncbi.nlm.nih.gov/gene/' + geneid, function (error, response
    , html) {
            var obj = { db : 'ncbi',
                        geneid : geneid,
                        official_symbol : [],
                        official_full_name : [],
                        see_related : [],
                        gene_symbol : [],
                        gene_description : [],
                        primary_source : [],
                        locus_tag : [],
                        gene_type : [],
                        rna_name : [],
                        refseq_status : [],
                        organism : [],
                        lineage : [],
                        also_known_as : [],
                        summary : [],
                        sequence : [],
                        location : [],
                        id_protein : []};
            if (typeof callback === 'undefined') {
                callback = save;
            }
            if (!error && response.statusCode == 200) {
                var $ = cheerio.load(html);
                var split = {};

                split.official_full_name = ', '
                split.see_related = '; ';
                split.lineage = '; ';
                split.also_known_as = '; ';

                $('#summaryDl').find('dt').each(function(i, elem) {
                    var aux = $(this).text().trim().toLowerCase().split(' ').join('_');

                    var aux2 = [];
                    aux2 = $(this).next().text().trim().split(split[aux]);
                    if(obj[aux])
                      obj[aux] = aux2;
                });

                $('.gc_cont').find('dt').each(function(i, elem) {
                    var aux = $(this).text().trim().toLowerCase().replace(':','').split
    (' ').join('_');
                    var aux2 = $(this).next().text().trim().split(split[aux]);
                    if(obj[aux])
                      obj[aux] = aux2;
                });
```

```
180.
181.                 callback(null, obj);
182.             } else {
183.                 callback(error, obj);
184.             }
185.         }).pipe(fq.createWriteStream('arquivos/html/ncbi/'+geneid+'.html'));
186.     },
187.
188.     save: function(error, obj) {
189.         var name;
190.         var path = 'arquivos/json/' + obj.db + '/';
191.         if (obj.db === 'ncbi') {
192.             name = obj.geneid;
193.         } else {
194.             name = obj.locus;
195.         }
196.         if(error) {
197.             console.log(error);
198.         } else {
199.             fq.writeFile(path + name + '.json', JSON.stringify(obj, null, 4), function(
        err) {
200.                 if(err) {
201.                   console.log(err);
202.                 }
203.             });
204.         }
205.     }
206. }
```

Apêndice B – Script para gerar o banco de dados

```
CREATE TABLE gene
(
  id serial NOT NULL,
  name character varying(20) NOT NULL,
  ncbigeneid integer DEFAULT 0,
  CONSTRAINT gene_pkey PRIMARY KEY (id),
  CONSTRAINT gene_name_key UNIQUE (name)
);

CREATE TABLE also_known_as
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT also_known_as_pkey PRIMARY KEY (id),
  CONSTRAINT also_known_as_info_key UNIQUE (info)
);

CREATE TABLE definition
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT definition_pkey PRIMARY KEY (id),
  CONSTRAINT definition_info_key UNIQUE (info)
);

CREATE TABLE description
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT description_pkey PRIMARY KEY (id),
  CONSTRAINT description_info_key UNIQUE (info)
);

CREATE TABLE gene_description
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT gene_description_pkey PRIMARY KEY (id),
  CONSTRAINT gene_description_info_key UNIQUE (info)
);

CREATE TABLE gene_model_type
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT gene_model_type_pkey PRIMARY KEY (id),
  CONSTRAINT gene_model_type_info_key UNIQUE (info)
);

CREATE TABLE gene_symbol
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT gene_symbol_pkey PRIMARY KEY (id),
  CONSTRAINT gene_symbol_info_key UNIQUE (info)
);

CREATE TABLE gene_type
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT gene_type_pkey PRIMARY KEY (id),
  CONSTRAINT gene_type_info_key UNIQUE (info)
```

```
);

CREATE TABLE id_protein
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT id_protein_pkey PRIMARY KEY (id),
  CONSTRAINT id_protein_info_key UNIQUE (info)
);

CREATE TABLE lineage
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT lineage_pkey PRIMARY KEY (id),
  CONSTRAINT lineage_info_key UNIQUE (info)
);

CREATE TABLE location
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT location_pkey PRIMARY KEY (id),
  CONSTRAINT location_info_key UNIQUE (info)
);

CREATE TABLE locus_tag
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT locus_tag_pkey PRIMARY KEY (id),
  CONSTRAINT locus_tag_info_key UNIQUE (info)
);

CREATE TABLE module
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT module_pkey PRIMARY KEY (id),
  CONSTRAINT module_info_key UNIQUE (info)
);

CREATE TABLE official_full_name
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT official_full_name_pkey PRIMARY KEY (id),
  CONSTRAINT official_full_name_info_key UNIQUE (info)
);

CREATE TABLE official_symbol
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT official_symbol_pkey PRIMARY KEY (id),
  CONSTRAINT official_symbol_info_key UNIQUE (info)
);

CREATE TABLE organism
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT organism_pkey PRIMARY KEY (id),
  CONSTRAINT organism_info_key UNIQUE (info)
);

```

```
CREATE TABLE other_dbs
(
  id serial NOT NULL,
  name text NOT NULL,
  info text NOT NULL,
  CONSTRAINT other_dbs_pkey PRIMARY KEY (id)
);

CREATE TABLE other_names
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT other_names_pkey PRIMARY KEY (id),
  CONSTRAINT other_names_info_key UNIQUE (info)
);

CREATE TABLE pathway
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT pathway_pkey PRIMARY KEY (id),
  CONSTRAINT pathway_info_key UNIQUE (info)
);

CREATE TABLE primary_source
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT primary_source_pkey PRIMARY KEY (id),
  CONSTRAINT primary_source_info_key UNIQUE (info)
);

CREATE TABLE refseq_status
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT refseq_status_pkey PRIMARY KEY (id),
  CONSTRAINT refseq_status_info_key UNIQUE (info)
);

CREATE TABLE rna_name
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT rna_name_pkey PRIMARY KEY (id),
  CONSTRAINT rna_name_info_key UNIQUE (info)
);

CREATE TABLE see_related
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT see_related_pkey PRIMARY KEY (id),
  CONSTRAINT see_related_info_key UNIQUE (info)
);

CREATE TABLE sequence
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT sequence_pkey PRIMARY KEY (id),
  CONSTRAINT sequence_info_key UNIQUE (info)
);

CREATE TABLE summary
(
```

```
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT summary_pkey PRIMARY KEY (id),
  CONSTRAINT summary_info_key UNIQUE (info)
);

CREATE TABLE located_in
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT located_in_pkey PRIMARY KEY (id),
  CONSTRAINT located_in_info_key UNIQUE (info)
);

CREATE TABLE functions_in
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT functions_in_pkey PRIMARY KEY (id),
  CONSTRAINT functions_in_info_key UNIQUE (info)
);

CREATE TABLE has
(
  id serial NOT NULL,
  info text NOT NULL,
  CONSTRAINT has_pkey PRIMARY KEY (id),
  CONSTRAINT has_info_key UNIQUE (info)
);

CREATE TABLE r_also_known_as
(
  id serial NOT NULL,
  gid integer NOT NULL,
  cid integer NOT NULL,
  CONSTRAINT r_also_known_as_pkey PRIMARY KEY (id),
  CONSTRAINT r_also_known_as_cid_fkey FOREIGN KEY (cid)
      REFERENCES also_known_as (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT r_also_known_as_gid_fkey FOREIGN KEY (gid)
      REFERENCES gene (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_definition
(
  id serial NOT NULL,
  gid integer NOT NULL,
  cid integer NOT NULL,
  CONSTRAINT r_definition_pkey PRIMARY KEY (id),
  CONSTRAINT r_definition_cid_fkey FOREIGN KEY (cid)
      REFERENCES definition (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT r_definition_gid_fkey FOREIGN KEY (gid)
      REFERENCES gene (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_description
(
  id serial NOT NULL,
  gid integer NOT NULL,
  cid integer NOT NULL,
  CONSTRAINT r_description_pkey PRIMARY KEY (id),
  CONSTRAINT r_description_cid_fkey FOREIGN KEY (cid)
      REFERENCES description (id) MATCH SIMPLE
```

```
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT r_description_gid_fkey FOREIGN KEY (gid)
        REFERENCES gene (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_gene_description
(
    id serial NOT NULL,
    gid integer NOT NULL,
    cid integer NOT NULL,
    CONSTRAINT r_gene_description_pkey PRIMARY KEY (id),
    CONSTRAINT r_gene_description_cid_fkey FOREIGN KEY (cid)
        REFERENCES gene_description (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT r_gene_description_gid_fkey FOREIGN KEY (gid)
        REFERENCES gene (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_gene_model_type
(
    id serial NOT NULL,
    gid integer NOT NULL,
    cid integer NOT NULL,
    CONSTRAINT r_gene_model_type_pkey PRIMARY KEY (id),
    CONSTRAINT r_gene_model_type_cid_fkey FOREIGN KEY (cid)
        REFERENCES gene_model_type (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT r_gene_model_type_gid_fkey FOREIGN KEY (gid)
        REFERENCES gene (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_gene_symbol
(
    id serial NOT NULL,
    gid integer NOT NULL,
    cid integer NOT NULL,
    CONSTRAINT r_gene_symbol_pkey PRIMARY KEY (id),
    CONSTRAINT r_gene_symbol_cid_fkey FOREIGN KEY (cid)
        REFERENCES gene_symbol (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT r_gene_symbol_gid_fkey FOREIGN KEY (gid)
        REFERENCES gene (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_gene_type
(
    id serial NOT NULL,
    gid integer NOT NULL,
    cid integer NOT NULL,
    CONSTRAINT r_gene_type_pkey PRIMARY KEY (id),
    CONSTRAINT r_gene_type_cid_fkey FOREIGN KEY (cid)
        REFERENCES gene_type (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT r_gene_type_gid_fkey FOREIGN KEY (gid)
        REFERENCES gene (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_id_protein
(
    id serial NOT NULL,
    gid integer NOT NULL,
```

```
    cid integer NOT NULL,
    CONSTRAINT r_id_protein_pkey PRIMARY KEY (id),
    CONSTRAINT r_id_protein_cid_fkey FOREIGN KEY (cid)
        REFERENCES id_protein (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT r_id_protein_gid_fkey FOREIGN KEY (gid)
        REFERENCES gene (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_lineage
(
    id serial NOT NULL,
    gid integer NOT NULL,
    cid integer NOT NULL,
    CONSTRAINT r_lineage_pkey PRIMARY KEY (id),
    CONSTRAINT r_lineage_cid_fkey FOREIGN KEY (cid)
        REFERENCES lineage (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT r_lineage_gid_fkey FOREIGN KEY (gid)
        REFERENCES gene (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_location
(
    id serial NOT NULL,
    gid integer NOT NULL,
    cid integer NOT NULL,
    CONSTRAINT r_location_pkey PRIMARY KEY (id),
    CONSTRAINT r_location_cid_fkey FOREIGN KEY (cid)
        REFERENCES location (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT r_location_gid_fkey FOREIGN KEY (gid)
        REFERENCES gene (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_locus_tag
(
    id serial NOT NULL,
    gid integer NOT NULL,
    cid integer NOT NULL,
    CONSTRAINT r_locus_tag_pkey PRIMARY KEY (id),
    CONSTRAINT r_locus_tag_cid_fkey FOREIGN KEY (cid)
        REFERENCES locus_tag (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT r_locus_tag_gid_fkey FOREIGN KEY (gid)
        REFERENCES gene (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_module
(
    id serial NOT NULL,
    gid integer NOT NULL,
    cid integer NOT NULL,
    CONSTRAINT r_module_pkey PRIMARY KEY (id),
    CONSTRAINT r_module_cid_fkey FOREIGN KEY (cid)
        REFERENCES module (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT r_module_gid_fkey FOREIGN KEY (gid)
        REFERENCES gene (id) MATCH SIMPLE
        ON UPDATE CASCADE ON DELETE CASCADE
);

```

```
CREATE TABLE r_official_full_name
(
  id serial NOT NULL,
  gid integer NOT NULL,
  cid integer NOT NULL,
  CONSTRAINT r_official_full_name_pkey PRIMARY KEY (id),
  CONSTRAINT r_official_full_name_cid_fkey FOREIGN KEY (cid)
      REFERENCES official_full_name (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT r_official_full_name_gid_fkey FOREIGN KEY (gid)
      REFERENCES gene (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_official_symbol
(
  id serial NOT NULL,
  gid integer NOT NULL,
  cid integer NOT NULL,
  CONSTRAINT r_official_symbol_pkey PRIMARY KEY (id),
  CONSTRAINT r_official_symbol_cid_fkey FOREIGN KEY (cid)
      REFERENCES official_symbol (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT r_official_symbol_gid_fkey FOREIGN KEY (gid)
      REFERENCES gene (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_organism
(
  id serial NOT NULL,
  gid integer NOT NULL,
  cid integer NOT NULL,
  CONSTRAINT r_organism_pkey PRIMARY KEY (id),
  CONSTRAINT r_organism_cid_fkey FOREIGN KEY (cid)
      REFERENCES organism (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT r_organism_gid_fkey FOREIGN KEY (gid)
      REFERENCES gene (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_other_dbs
(
  id serial NOT NULL,
  gid integer NOT NULL,
  cid integer NOT NULL,
  CONSTRAINT r_other_dbs_pkey PRIMARY KEY (id),
  CONSTRAINT r_other_dbs_cid_fkey FOREIGN KEY (cid)
      REFERENCES other_dbs (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT r_other_dbs_gid_fkey FOREIGN KEY (gid)
      REFERENCES gene (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_other_names
(
  id serial NOT NULL,
  gid integer NOT NULL,
  cid integer NOT NULL,
  CONSTRAINT r_other_names_pkey PRIMARY KEY (id),
  CONSTRAINT r_other_names_cid_fkey FOREIGN KEY (cid)
      REFERENCES other_names (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT r_other_names_gid_fkey FOREIGN KEY (gid)
```

```
460.        REFERENCES gene (id) MATCH SIMPLE
461.        ON UPDATE CASCADE ON DELETE CASCADE
462.  );
463.
464.  CREATE TABLE r_pathway
465.  (
466.    id serial NOT NULL,
467.    gid integer NOT NULL,
468.    cid integer NOT NULL,
469.    CONSTRAINT r_pathway_pkey PRIMARY KEY (id),
470.    CONSTRAINT r_pathway_cid_fkey FOREIGN KEY (cid)
471.        REFERENCES pathway (id) MATCH SIMPLE
472.        ON UPDATE CASCADE ON DELETE CASCADE,
473.    CONSTRAINT r_pathway_gid_fkey FOREIGN KEY (gid)
474.        REFERENCES gene (id) MATCH SIMPLE
475.        ON UPDATE CASCADE ON DELETE CASCADE
476.  );
477.
478.  CREATE TABLE r_primary_source
479.  (
480.    id serial NOT NULL,
481.    gid integer NOT NULL,
482.    cid integer NOT NULL,
483.    CONSTRAINT r_primary_source_pkey PRIMARY KEY (id),
484.    CONSTRAINT r_primary_source_cid_fkey FOREIGN KEY (cid)
485.        REFERENCES primary_source (id) MATCH SIMPLE
486.        ON UPDATE CASCADE ON DELETE CASCADE,
487.    CONSTRAINT r_primary_source_gid_fkey FOREIGN KEY (gid)
488.        REFERENCES gene (id) MATCH SIMPLE
489.        ON UPDATE CASCADE ON DELETE CASCADE
490.  );
491.
492.  CREATE TABLE r_refseq_status
493.  (
494.    id serial NOT NULL,
495.    gid integer NOT NULL,
496.    cid integer NOT NULL,
497.    CONSTRAINT r_refseq_status_pkey PRIMARY KEY (id),
498.    CONSTRAINT r_refseq_status_cid_fkey FOREIGN KEY (cid)
499.        REFERENCES refseq_status (id) MATCH SIMPLE
500.        ON UPDATE CASCADE ON DELETE CASCADE,
501.    CONSTRAINT r_refseq_status_gid_fkey FOREIGN KEY (gid)
502.        REFERENCES gene (id) MATCH SIMPLE
503.        ON UPDATE CASCADE ON DELETE CASCADE
504.  );
505.
506.  CREATE TABLE r_rna_name
507.  (
508.    id serial NOT NULL,
509.    gid integer NOT NULL,
510.    cid integer NOT NULL,
511.    CONSTRAINT r_rna_name_pkey PRIMARY KEY (id),
512.    CONSTRAINT r_rna_name_cid_fkey FOREIGN KEY (cid)
513.        REFERENCES rna_name (id) MATCH SIMPLE
514.        ON UPDATE CASCADE ON DELETE CASCADE,
515.    CONSTRAINT r_rna_name_gid_fkey FOREIGN KEY (gid)
516.        REFERENCES gene (id) MATCH SIMPLE
517.        ON UPDATE CASCADE ON DELETE CASCADE
518.  );
519.
520.  CREATE TABLE r_see_related
521.  (
522.    id serial NOT NULL,
523.    gid integer NOT NULL,
524.    cid integer NOT NULL,
525.    CONSTRAINT r_see_related_pkey PRIMARY KEY (id),
```

```
  CONSTRAINT r_see_related_cid_fkey FOREIGN KEY (cid)
      REFERENCES see_related (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT r_see_related_gid_fkey FOREIGN KEY (gid)
      REFERENCES gene (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_sequence
(
  id serial NOT NULL,
  gid integer NOT NULL,
  cid integer NOT NULL,
  CONSTRAINT r_sequence_pkey PRIMARY KEY (id),
  CONSTRAINT r_sequence_cid_fkey FOREIGN KEY (cid)
      REFERENCES sequence (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT r_sequence_gid_fkey FOREIGN KEY (gid)
      REFERENCES gene (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_summary
(
  id serial NOT NULL,
  gid integer NOT NULL,
  cid integer NOT NULL,
  CONSTRAINT r_summary_pkey PRIMARY KEY (id),
  CONSTRAINT r_summary_cid_fkey FOREIGN KEY (cid)
      REFERENCES summary (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT r_summary_gid_fkey FOREIGN KEY (gid)
      REFERENCES gene (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_located_in
(
  id serial NOT NULL,
  gid integer NOT NULL,
  cid integer NOT NULL,
  CONSTRAINT r_located_in_pkey PRIMARY KEY (id),
  CONSTRAINT r_located_in_cid_fkey FOREIGN KEY (cid)
      REFERENCES located_in (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT r_located_in_gid_fkey FOREIGN KEY (gid)
      REFERENCES gene (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_functions_in
(
  id serial NOT NULL,
  gid integer NOT NULL,
  cid integer NOT NULL,
  CONSTRAINT r_functions_in_pkey PRIMARY KEY (id),
  CONSTRAINT r_functions_in_cid_fkey FOREIGN KEY (cid)
      REFERENCES functions_in (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT r_functions_in_gid_fkey FOREIGN KEY (gid)
      REFERENCES gene (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE r_has
(
```

```
592.    id serial NOT NULL,
593.    gid integer NOT NULL,
594.    cid integer NOT NULL,
595.    CONSTRAINT r_has_pkey PRIMARY KEY (id),
596.    CONSTRAINT r_has_cid_fkey FOREIGN KEY (cid)
597.        REFERENCES has (id) MATCH SIMPLE
598.        ON UPDATE CASCADE ON DELETE CASCADE,
599.    CONSTRAINT r_has_gid_fkey FOREIGN KEY (gid)
600.        REFERENCES gene (id) MATCH SIMPLE
601.        ON UPDATE CASCADE ON DELETE CASCADE
602. );
```

Apêndice C – Script para gerar as views

```
1.  CREATE VIEW vdescription AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS
     locus FROM description i, r_description r, gene g WHERE i.id = r.cid AND g.id = r.gi
    d;
2.  CREATE VIEW vgene_model_type AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.nam
    e AS locus FROM gene_model_type i, r_gene_model_type r, gene g WHERE i.id = r.cid AND
     g.id = r.gid;
3.  CREATE VIEW vother_names AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS
     locus FROM other_names i, r_other_names r, gene g WHERE i.id = r.cid AND g.id = r.gi
    d;
4.  CREATE VIEW vlocated_in AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS
    locus FROM located_in i, r_located_in r, gene g WHERE i.id = r.cid AND g.id = r.gid;

5.  CREATE VIEW vfunctions_in AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name A
    S locus FROM functions_in i, r_functions_in r, gene g WHERE i.id = r.cid AND g.id = r
    .gid;
6.  CREATE VIEW vhas AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS locus F
    ROM has i, r_has r, gene g WHERE i.id = r.cid AND g.id = r.gid;
7.  CREATE VIEW vdefinition AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS
    locus FROM definition i, r_definition r, gene g WHERE i.id = r.cid AND g.id = r.gid;

8.  CREATE VIEW vmodule AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS locu
    s FROM module i, r_module r, gene g WHERE i.id = r.cid AND g.id = r.gid;
9.  CREATE VIEW vother_dbs AS SELECT i.name AS db, i.info AS info, i.id AS id, g.id AS gi
    d, g.name AS locus FROM other_dbs i, r_other_dbs r, gene g WHERE i.id = r.cid AND g.i
    d = r.gid;
10. CREATE VIEW vpathway AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS loc
    us FROM pathway i, r_pathway r, gene g WHERE i.id = r.cid AND g.id = r.gid;
11. CREATE VIEW vofficial_symbol AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.nam
    e AS locus FROM official_symbol i, r_official_symbol r, gene g WHERE i.id = r.cid AND
     g.id = r.gid;
12. CREATE VIEW vofficial_full_name AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.
    name AS locus FROM official_full_name i, r_official_full_name r, gene g WHERE i.id =
    r.cid AND g.id = r.gid;
13. CREATE VIEW vsee_related AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS
     locus FROM see_related i, r_see_related r, gene g WHERE i.id = r.cid AND g.id = r.gi
    d;
14. CREATE VIEW vgene_symbol AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS
     locus FROM gene_symbol i, r_gene_symbol r, gene g WHERE i.id = r.cid AND g.id = r.gi
    d;
15. CREATE VIEW vgene_description AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.na
    me AS locus FROM gene_description i, r_gene_description r, gene g WHERE i.id = r.cid
    AND g.id = r.gid;
16. CREATE VIEW vprimary_source AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name
     AS locus FROM primary_source i, r_primary_source r, gene g WHERE i.id = r.cid AND g.
    id = r.gid;
17. CREATE VIEW vlocus_tag AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS l
    ocus FROM locus_tag i, r_locus_tag r, gene g WHERE i.id = r.cid AND g.id = r.gid;
18. CREATE VIEW vgene_type AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS l
    ocus FROM gene_type i, r_gene_type r, gene g WHERE i.id = r.cid AND g.id = r.gid;
19. CREATE VIEW vrna_name AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS lo
    cus FROM rna_name i, r_rna_name r, gene g WHERE i.id = r.cid AND g.id = r.gid;
20. CREATE VIEW vrefseq_status AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name
    AS locus FROM refseq_status i, r_refseq_status r, gene g WHERE i.id = r.cid AND g.id
    = r.gid;
21. CREATE VIEW vorganism AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS lo
    cus FROM organism i, r_organism r, gene g WHERE i.id = r.cid AND g.id = r.gid;
22. CREATE VIEW vlineage AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS loc
    us FROM lineage i, r_lineage r, gene g WHERE i.id = r.cid AND g.id = r.gid;
23. CREATE VIEW valso_known_as AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name
    AS locus FROM also_known_as i, r_also_known_as r, gene g WHERE i.id = r.cid AND g.id
    = r.gid;
```

```
24.  CREATE VIEW vsummary AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS loc
     us FROM summary i, r_summary r, gene g WHERE i.id = r.cid AND g.id = r.gid;
25.  CREATE VIEW vsequence AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS lo
     cus FROM sequence i, r_sequence r, gene g WHERE i.id = r.cid AND g.id = r.gid;
26.  CREATE VIEW vlocation AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS lo
     cus FROM location i, r_location r, gene g WHERE i.id = r.cid AND g.id = r.gid;
27.  CREATE VIEW vid_protein AS SELECT i.info AS info, i.id AS id, g.id AS gid, g.name AS
     locus FROM id_protein i, r_id_protein r, gene g WHERE i.id = r.cid AND g.id = r.gid;
```

Apêndice D – Script para inserção das informações no banco

```
1.    var fs = require('fs');
2.    var FileQueue = require('filequeue');
3.    var fq = new FileQueue(100);
4.    var anyDB = require('any-db');
5.    var conn = anyDB.createPool('postgres://postgres:testebio@localhost/teste', {min: 1,
      max: 1});
6.
7.    function escape (str) {
8.        return str.replace(/[\0\x08\x09\x1a\n\r"'\\\%]/g, function (char) {
9.            switch (char) {
10.               case "\0":
11.                   return "\\0";
12.               case "\x08":
13.                   return "\\b";
14.               case "\x09":
15.                   return "\\t";
16.               case "\x1a":
17.                   return "\\z";
18.               case "\n":
19.                   return "\\n";
20.               case "\r":
21.                   return "\\r";
22.               case "\"":
23.               case "\\":
24.               case "%":
25.                   return "\\"+char;
26.           }
27.       });
28.   }
29.
30.   var tair = fs.readdirSync(__dirname + '/arquivos/json/tair/');
31.   var kegg = fs.readdirSync(__dirname + '/arquivos/json/kegg/');
32.   var ncbi = fs.readdirSync(__dirname + '/arquivos/json/ncbi/');
33.   var files = {tair: tair, kegg: kegg, ncbi: ncbi}
34.
35.   function addInfo (n1, n2, ni) {
36.       var infoArray = [{i: 'description', db: 'tair'},
37.                        {i: 'gene_model_type', db: 'tair'},
38.                        {i: 'other_names', db: 'tair'},
39.                        {i: 'located_in', db: 'tair'},
40.                        {i: 'functions_in', db: 'tair'},
41.                        {i: 'has', db: 'tair'},
42.                        {i: 'definition', db: 'kegg'},
43.                        {i: 'module', db: 'kegg'},
44.                        {i: 'other_dbs', db: 'kegg'},
45.                        {i: 'pathway', db: 'kegg'},
46.                        {i: 'official_symbol', db: 'ncbi'},
47.                        {i: 'official_full_name', db: 'ncbi'},
48.                        {i: 'see_related', db: 'ncbi'},
49.                        {i: 'gene_symbol', db: 'ncbi'},
50.                        {i: 'gene_description', db: 'ncbi'},
51.                        {i: 'primary_source', db: 'ncbi'},
52.                        {i: 'locus_tag', db: 'ncbi'},
53.                        {i: 'gene_type', db: 'ncbi'},
54.                        {i: 'rna_name', db: 'ncbi'},
55.                        {i: 'refseq_status', db: 'ncbi'},
56.                        {i: 'organism', db: 'ncbi'},
57.                        {i: 'lineage', db: 'ncbi'},
58.                        {i: 'also_known_as', db: 'ncbi'},
59.                        {i: 'summary', db: 'ncbi'},
60.                        {i: 'sequence', db: 'ncbi'},
61.                        {i: 'location', db: 'ncbi'},
```

```
                    {i: 'id_protein', db: 'ncbi'}];
    var info = infoArray[ni].i;
    var db = infoArray[ni].db;
    var infoPrefixo = '';
    var relPrefixo = 'r_';
    var query1 = '';
    var query2 = '';

    var data = fs.readFileSync(__dirname + '/arquivos/json/' + db + '/' + files[db][n1]);
    data = JSON.parse(data);

    var where = 'WHERE name = \'' + data.locus + '\'';
    if (db === 'ncbi') {
        where = 'WHERE ncbigeneid = \'' + data.geneid + '\'';
    }
    var x = data[info];

    if (n2 == x.length) {
        if (++n1 < files[db].length) {
            if (x.length == 0)
            do {

                d = fs.readFileSync(__dirname + '/arquivos/json/' + db + '/' + files[db][++n1]);
                var d = JSON.parse(d);
                var z = d[info].length;
                if (z > 0)
                    break;

            } while (n1 < files[db].length-1);
            addInfo(n1, 0, ni);
        } else {
            console.log(info + ' Populado!');
            if (++ni < infoArray.length)
                addInfo(0, 0, ni);
        }
    } else {
        if (info === 'other_dbs') {
            query1 = 'SELECT id FROM ' + info + ' WHERE info = \'' + escape(x[n2].id) + '\' AND name = \'' + escape(x[n2].name) + '\'';
            query2 = 'INSERT INTO ' + info + ' (info, name) VALUES(\'' + escape(x[n2].id) + '\',\'' + escape(x[n2].name) + '\') RETURNING id';
        } else {
            query1 = 'SELECT id FROM ' + infoPrefixo + info + ' WHERE info = \'' + escape(x[n2]) + '\'';
            query2 = 'INSERT INTO ' + infoPrefixo + info + ' (info) VALUES(\'' + escape(x[n2]) + '\') RETURNING id';
        }
        conn.query(query1, function (err, res) {
            if(err) {
                console.log('1: '+err);
            } else {
                if(res.rows.length == 0) {
                    conn.query(query2, function (err, res) {
                        if(err) {
                            console.log('2: '+err);
                        } else {
                            conn.query('INSERT INTO ' + relPrefixo + info + ' (gid, cid) VALUES ((SELECT id FROM gene ' + where + '), \'' + res.rows[0].id + '\')', function (err, res) {
                                if(err)
                                    console.log('3: '+err);
                            });
                            n2++;
                            addInfo(n1, n2, ni);
```

```
                            }
                        });
                    } else {
                        conn.query('SELECT id FROM ' + relPrefixo + info + ' WHERE gid = (SELECT id FROM gene ' + where + ') AND cid = \'' + res.rows[0].id + '\'', function (err, resp) {
                            if(err)
                                console.log('4: '+err);
                            if (resp.rows.length == 0)
                                conn.query('INSERT INTO ' + relPrefixo + info + ' (gid, cid) VALUES ((SELECT id FROM gene ' + where + '), \'' + res.rows[0].id + '\')', function (err, res) {
                                    if(err)
                                        console.log('5: '+err);
                                });
                            n2++;
                            addInfo(n1, n2, ni);
                        });
                    }
                }
            });
        }
}

addInfo(0, 0, 0);
```

Apêndice E - Artigo

A feature selection approach for evaluate the inference of GRNs through biological data integration - A case study on *A. thaliana*

Fábio F. R. Vicente[1,2], Euler Menezes[1], Gabriel Rubino[1], Juliana de Oliveira[3] and Fabrício M. Lopes[1]

[1] **F**ederal University of Technology, Paraná, Brazil.
fabiofernandes,fabricio{@utfpr.edu.br}
[2] Institute of Mathematics and Statistics,University of São Paulo, São Paulo, Brazil.
[3] Department of Biological Sciences Faculty of Sciences and Letters of Assis - FCLA University of São Paulo State - UNESP Av. Dom Antonio, 2100, Parque Universitrio 19806-900 - Assis, São Paulo, Brazil.
juliana@assis.unesp.br

Abstract. The inference of gene regulatory networks (GRNs) from expression profiles is a great challenge in bioinformatics due to the curse of dimensionality. For this reason, several methods that perform data integration have been developed to reduce the estimation error of the inference. However, it is not completely formulated how to use each type of biological information available. This work address this issue by proposing feature selection approach in order to integrate biological data and evaluate three types of biological information regarding their effect on the similarity of inferred GRNs. The proposed feature selection method is based on sequential forward floating selection (SFFS) search algorithm and the mean conditional entropy (MCE) as criterion function. An expression dataset was built as an additional contribution of this work containing 22746 genes and 1206 experiments regarding *A. thaliana*. The experimental results achieve 39% of GRNs improvement in average when compared to non-use of biological data integration. Besides, the results showed that the improvement is associated to a specific type of biological information: the cellular localization, which is a valuable and information for the development of new experiments and indicates an important insight for investigation.

Keywords: Gene regulatory networks, feature selection, data integration, bioinformatics, *Arabidopsis thaliana*.

1 Introduction

One of the main challenges in bioinformatics is to perform the reverse engineering of GRNs (gene regulatory networks) from expression profiles [2]. Since the genes and their corresponding proteins are only *part* of a whole biological system, it is important to discover the network of interactions between the cell components to better understand how the systems work. Furthermore, such knowledge can be useful to perform intervention and control of the system[23]. The development of

techniques such as DNA microarrays, SAGE and RNA-Seq [29] allowed studying the dynamic of the cell in a scale that was unfeasible with the previous methods. Since the interaction between the genes works as a network that produces the observed expression, the idea is to perform a reverse engineering in order to recover the network from the expression profiles.

The search space of possible networks is huge. Thus, the inference can be modeled as a feature selection problem, where a criterion function is used to evaluate a subset of predictor genes (features) that better classify the state of a given target gene based on the expression values [17, 16]. In this way, the inference consists in searching for a subset of predictors for each target.

A GRN can be modeled as Bayesian Networks [12], Boolean Networks [8], Probabilistic Boolean Networks (PBN) [25] or Probabilistic Genetic Network (PGN) [3]. Also, distinct *criterion function* can be used for inference. Some criterion function as Pearson correlation [26] are limited to the evaluation of pairs of genes and the combinatorial regulation of multiple predictors cannot be assessed. The Coefficient of Determination (CoD) [13] is not limited to a number of variables and allows to model the multivariate nature of the regulation. Other criterion functions are based on information theory such as Mutual Information [22], Tsallis Entropy [19] and Mean Conditional Entropy (MCE) [18].

One of the main limitations of the inference is the high dimensionality. For this reason, it is important the development of new methods that take into account other *information* than the expression profiles in order to reduce the intrinsic estimation error of the inference and to recover plausible biological networks.

Distinct approaches have been proposed to implement data integration in GNs. Some works evaluate the data integration in biological networks [21, 28, 20]. Other methods perform a clustering analysis to identify groups of genes related to some particular property [7]. Other methods include known information about the topological properties of the network such as degree of connectivity, average path length and other local and global characteristics [16, 27].

However, a methodology to define how to select the biological information in order to reduce error on inference is still not completely formulated. Moreover, the study of distinct biological information can reveal the contribution of each one in the inference and favor the discovery of new biological knowledge such as biological information that are decisive to characterize the network together with the system dynamic. This work proposes a feature selection approach in order to evaluate a GRNs inference model based on a criterion function that encodes multivalued biological features applied on the *A. thaliana* organism.

2 Materials and Methods

2.1 Gold standard network

The gold standard network is the directed graph with the known physical direct interactions between each predictor (TF) and their target genes. The *Arabidopsis thaliana* gold standard network (AtRegNet) was obtained from AGRIS (The Arabidopsis Gene Regulatory Information Server) [30] on which each interaction

is verified in at least one experimental approach. The graph has 8154 genes of which 67 are predictors and 8131 are target genes, comprising 11481 edges.

Expression data All expression data were obtained from GEO [4] and only samples of the platform GPL198 (*Affymetrix Arabidopsis ATH1 Genome Array*) were selected. The chip contains 22810 *probe sets* that were mapped to genes through annotation data from TAIR [15], Gene Ontology [1] and TIGR [5]. A probe set is a collection of sequences (probes) used to identify a gene sequence and to measure its expression.

The GEO files containing the expression samples were obtained through the R Bioconductor , GEOquery and Biobase [6]. The files in SOFT format were acquired in three types: GDS (GEO Dataset, 13 files), GSE (GEO Series, 58 files) and GSM (GEO Samples, 109 files), each one with a set of expression experiments. The files were preprocessed to obtain the expression values and to filter out samples with missing data. Thus the 180 files resulted on a table with 22746 genes and 1206 experiments.

Biological Information A set of features associated to each probe set in the expression data where obtained from TAIR, KEGG [14] and NCBI [11]. For each probe set a corresponding *locus identifier* (TAIR) and a *NCBI gene id* were obtained. Then, the biological information associated to the NCBI gene id were obtained from the NCBI and features associated to the locus identifier were obtained from TAIR and KEGG. The dataset contains 23593 annotated elements and 15 features related to several biological aspects.

2.2 GRNs Inference

To infer the GRN it is necessary determine the best subset of predictors for each target gene. Thus, in a dataset with n transcription factors there are 2^n possible subset of predictors for each gene. Since the search space is huge, the sequential forward floating selection (SFFS)[24] algorithm was adopted to search for the best solution. The criterion function evaluates each subset by taking into account both the expression values and the biological information on the dataset. The expression value part is evaluated computing the mean conditional entropy (MCE) [18]. The MCE is computed as the average of the entropies of a target gene Y given the values of the predictors X (Eq. 1).

$$H(Y|X) = \sum_{x \in X} H(Y|x)P(x). \tag{1}$$

The MCE take into account only the expression values. Because the small number of samples it is common the occurrence of many ties between distinct set of predictors. Thus, only the expression data is commonly insufficient to determine a suitable solution. For this reason, the proposed criterion function includes a term related to another type of feature, i.e., a biological information.

$$F(Y, X) = \alpha[H(Y|X)] + \beta[D(Y, X)], \tag{2}$$

where $D(Y, X)$ is equals to 1 when the biological information of the gene Y is equals to X in the biological data set D. The parameters α and β defines the weight of each type of information where $\beta \in (0, 1)$ and $\alpha = 1 - \beta$. Thus, with the proposed feature selection approach the SFFS will search for the subset of predictors that not just minimize the entropy, but also for those that are coherent to the other biological aspects of the target gene.

3 Results and discussion

Data preprocessing The expression data are represented in a table with the variables (genes) in rows and the expression samples $s(i)$ in columns. The expression values of the genes of each sample $s(i)$, $s(i) \in \mathbb{R}$, were normalized by the *normal transformation*, defined as follows:

$$\eta[s(i)] = \frac{s(i) - E[s(i)]}{\sigma[s(i)]}, \tag{3}$$

where $\sigma[s(i)]$ is the standard deviation of $s(i)$ and $E[s(i)]$ is the expectation of $s(i)$. The normalized data where discretized into three levels {-1,0,+1} through a threshold mapping

$$s(i) = \begin{cases} -1 & \alpha < l \\ 0 & l \leq \alpha \leq h \\ +1 & \alpha > h \end{cases} \tag{4}$$

where, α is the expression value of the sample s, of the gene g.

Biological features The annotation of genes on public databases makes available several types of information, which refer to distinct aspects. They can be descriptive, some times adopting conventional terms to classify the gene as belonging to a category, for example, to a specific biological pathway, a known function, to cite but a few. Thus, in this work were adopted three features related to *physical* aspects and to the *activity* of the gene: (i) cellular location, (ii) pathway and (iii) function. Then, since all annotated features into the databases are attributes of type *nominal*, the value of each feature was indexed by an integer value resulting on a dataset with 20817 unique annotated pairs (*probe set, gene locus*) Table 1.

Table 1: Biological information available on public databases.

Database	**Feature**	**Number of distinct values**
TAIR	function	121
TAIR	located in	6
KEGG	pathway	128

Intersection between datasets To evaluate the proposed approach, the subset of genes in the intersection of the three datasets (gold standard network, expression and biological information) was adopted, resulting on 5974 Gene locus (Table 2).

Table 2: The validation dataset comprises 5974 genes.

Dataset	Description	Quantity
Gold standard network	Edges	8589
Expression data	Samples	1206
Biological features	Features	3

Evaluation of the inference In order to measure the effectiveness of the proposed approach it was adopted the *Similarity* between the gold standard and the inferred network, which was presented by [9] and is widely used for validation of methods of GNs inference. The validation is based on a confusion matrix, as described in Table 3. The *Similarity* is computed as presented in (Eq.5) and the normalized value relative to the inference based only on expression data($\alpha = 1$), defined as follows:

Table 3: Confusion matrix. TP = true positive, FN = false negative, FP = false positive, TN = true negative.

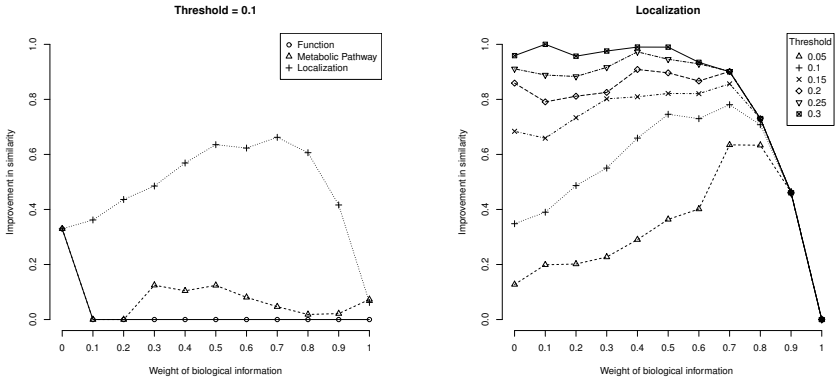
Edge	Inferred	Not Inferred
Present	TP	FN
Absent	FP	TN

$$\text{Similarity} = \sqrt{\text{precision} \times \text{recall}} = \sqrt{\frac{TP}{TP+FP} \times \frac{TP}{TP+FN}}. \quad (5)$$

The experimental results have presented distinct performance for the three types of biological information (Figure 1(a)). In particular, the *localization* improves the similarity while *pathway* and *function* increased the error of the inference. For *cellular localization*, the curve increases from β varying from 0.1 to 0.7 and decreases for values greater than 0.7. This indicates that the expression combined to cellular localization can improve the similarity and that the use of only one type of data can limit the inference. The improvement varies according to the threshold, being higher for small values. The average improvement when the same weight is given to both expression and biological information (i.e. $\beta = 0.5$) is 39.1%.

The other two biological information presented a negative effect in similarity. A possible explanation is that the gold standard regulatory network refers to *physical* interactions and metabolic pathway and gene function refers to other type of relationship. The localization refers to a physical space on the cell where genes acts, metabolic pathway is more generic than localization and refers to the biological pathway the genes are related (not necessarily the genes are in the same local or interacts physically each other) and, an even more generic than metabolic pathway is the gene function. Thus, several proteins can have the same molecular function or acts in common metabolic pathways and can be an explanation for the decreasing in similarity. This points to the importance of the correct feature selection that can reduce the estimation error, specially in this type of multilevel data integration.

Figure 1(b) shows the improvement in similarity. Results show the similarity increases more and faster for lower thresholds and that both expression and



(a) Biological information

(b) Similarity for different criterion function thresholds

Fig. 1: Evaluation of the data integration of three types of biological information: Function, Metabolic pathway and Cellular localization. Weight = 0 means only expression. Weight = 1 means only biological information.

localization are important. For weight of biological information over 0.7 the similarity decreases faster despite of the threshold.

Another important issue is the validation of GNs inference. Here, we adopted the validation regarding to a know network, which useful to evaluate the *inferential* perspective as pointed by [10]. The validation of methods of data integration is still an problem to be solved in GNs inference.

4 Conclusions

This work presented a feature selection approach for integration of distinct datasets based on expression data and biological information for the inference of gene regulatory networks, which is based on SFFS search algorithm and MCE criterion function. An expression dataset was built as an additional contribution of this work containing 22746 genes and 1206 experiments regarding *A. thaliana*. The dataset was composed with a gold standard network, expression data and biological features were assembled and the proposed approach was applied on the composed dataset. The results showed the increasing on the similarity (average 39%) of the recovered network when the criterion function is based on cellular localization. Also, the results showed that the performance is better when expression and cellular localization are combined instead of the use of each one only, which is a valuable information for the development of new experiments and indicates an important insight for investigation.

The evaluation of GNs inference is commonly performed based on the know links of a gold-standard network. However, other biological information could be also used to validate the inference methods. Thus, as future work it is suggest the investigation of validation measures that take into account distinct biological data. Also, the inclusion of more types of biological information and to perform the proposed approach using these biological information can be considered as a

further work. Moreover, would be also important to evaluate the best set of the parameters (weight of biological information and the threshold of the criterion function).

Acknowledgements

This work was supported by FAPESP grant 2011/50761-2, UTFPR, CNPq, Fundação Araucária, CAPES and NAP eScience - PRP - USP.

References

1. Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D., Butler, H., Cherry, J.M., Davis, A.P., Dolinski, K., Dwight, S.S., Eppig, J.T., Harris, M.A., Hill, D.P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J.C., Richardson, J.E., Ringwald, M., Rubin, G.M., Sherlock, G.: Gene ontology: tool for the unification of biology. Nat Genet 25(1), 25–29 (2000)
2. Baralla, A., Mentzen, W.I., de la Fuente, A.: Inferring gene networks: dream or nightmare? Annals of the New York Academy of Sciences 1158, 246–56 (Mar 2009)
3. Barrera, J., Cesar-Jr, R.M., Martins-Jr, D.C., Vencio, R.Z.N., Merino, E.F., Yamamoto, M.M., Leonardi, F.G., Pereira, C.A.B., Portillo, H.A.: Constructing probabilistic genetic networks of *Plasmodium falciparum*, from dynamical expression signals of the intraerythrocytic development cycle. In: McConnell, P., Lin, S.M., Hurban, P. (eds.) Meth. of Microarray Data Analysis, pp. 11–26. Springer (2007)
4. Barrett, T., Wilhite, S.E., Ledoux, P., Evangelista, C., Kim, I.F., Tomashevsky, M., Marshall, K.a., Phillippy, K.H., Sherman, P.M., Holko, M., Yefanov, A., Lee, H., Zhang, N., Robertson, C.L., Serova, N., Davis, S., Soboleva, A.: NCBI GEO: Archive for functional genomics data sets - Update. NAR 41(D1), 991–995 (2013)
5. Childs, K.L., Hamilton, J.P., Zhu, W., Ly, E., Cheung, F., Wu, H., Rabinowicz, P.D., Town, C.D., Buell, C.R., Chan, A.P.: The tigr plant transcript assemblies database. Nucleic Acids Research 35(suppl 1), D846–D851 (2007)
6. Davis, S., Meltzer, P.: Geoquery: a bridge between the gene expression omnibus (geo) and bioconductor. Bioinformatics 14, 1846–1847 (2007)
7. De Haan, J., Piek, E., van Schaik, R., de Vlieg, J., Bauerschmidt, S., Buydens, L., Wehrens, R.: Integrating gene expression and go classification for pca by preclustering. BMC Bioinformatics 11(1), 158 (2010)
8. D'haeseleer, P., Liang, S., Somogyi, R.: Genetic network inference: from co-expression clustering to reverse engineering. Bioinformatics 16(8), 707–726 (2000)
9. Dougherty, E.R.: Validation of inference procedures for gene regulatory networks. Current Genomics 8(6), 351–359 (2007)
10. Dougherty, E.R.: Validation of gene regulatory networks: scientific and inferential. Briefings in Bioinformatics 12(3), 245–252 (2011)
11. Edgar, R., Domrachev, M., Lash, A.E.: Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. Nucleic acids research 30(1), 207–210 (2002)
12. Friedman, N., Linial, M., Nachman, I., Pe'er, D.: Using bayesian networks to analyze expression data. Journal of Computational Biology 7(3-4), 601–620 (2000)
13. Hashimoto, R.F., Kim, S., Shmulevich, I., Zhang, W., Bittner, M.L., Dougherty, E.R.: Growing genetic regulatory networks from seed genes. Bioinformatics 20(8), 1241–1247 (May 2004)
14. Kanehisa, M., Goto, S.: KEGG: Kyoto Encyclopedia of Genes and Genomes. Nucleic Acids Research 28(1), 27–30 (2000)

15. Lamesch, P., Berardini, T.Z., Li, D., Swarbreck, D., Wilks, C., Sasidharan, R., Muller, R., Dreher, K., Alexander, D.L., Garcia-Hernandez, M., Karthikeyan, A.S., Lee, C.H., Nelson, W.D., Ploetz, L., Singh, S., Wensel, A., Huala, E.: The arabidopsis information resource (TAIR): improved gene annotation and new tools. NAR (2011)
16. Lopes, F.M., Jr., D.C.M., Barrera, J., Jr., R.M.C.: A feature selection technique for inference of graphs from their known topological properties: Revealing scale-free gene regulatory networks. Information Sciences 272(0), 1–15 (2014)
17. Lopes, F.M., Martins-Jr, D.C., Barrera, J., Cesar-Jr, R.M.: SFFS-MR: a floating search strategy for GRNs inference. In: Pattern Recognition in Bioinformatics, Proceedings. LNCS, vol. 6282, pp. 407–418. Springer Berlin / Heidelberg (2010)
18. Lopes, F.M., Martins-Jr, D.C., Cesar-Jr, R.M.: Feature selection environment for genomic applications. BMC Bioinformatics 9(1), 451 (October 2008)
19. Lopes, F.M., de Oliveira, E.A., Cesar-Jr, R.M.: Inference of gene regulatory networks from time series by Tsallis entropies. BMC Systems Biology 5(1), 61 (2011)
20. Lopes, F.M., Ray, S.S., Hashimoto, R.F., Jr., R.M.C.: Entropic biological score: a cell cycle investigation for GRNs inference. Gene 541(2), 129–137 (2014)
21. Lu, L.J., Xia, Y., Paccanaro, A., Yu, H., Gerstein, M.: Assessing the limits of genomic data integration for predicting protein networks. Gen. Res. 15(7), 945–53 (2005)
22. Margolin, A., Basso, K.N., Wiggins, C., Stolovitzky, G., Favera, R., Califano, A.: ARACNE: An algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context. BMC Bioinformatics 7(Suppl 1), S7 (2006)
23. Pavlopoulos, G.A., Secrier, M., Moschopoulos, C.N., Soldatos, T.G., Kossida, S., Aerts, J., Schneider, R., Bagos, P.G.: Using graph theory to analyze biological networks. BioData mining 4(1), 10 (jan 2011)
24. Pudil, P., Novovičová, J., Kittler, J.: Floating search methods in feature-selection. Pattern Recognition Letters 15(11), 1119–1125 (November 1994)
25. Shmulevich, I., Dougherty, E.R., Kim, S., Zhang, W.: Probabilistic boolean networks: a rule-based uncertainty model for gene regulatory networks. Bioinformatics 18(2), 261–274 (2002)
26. Stuart, J.M., Segal, E., Koller, D., Kim, S.K.: A gene-coexpression network for global discovery of conserved genetic modules. Science 302(5643), 249–255 (2003)
27. Vicente, F.F.R., Lopes, F.M.: SFFS-WS: A feature selection algorithm exploring the small-world properties of GNs. In: Pattern Recognition in Bioinformatics, Proceedings. LNCS, vol. 8626, pp. 60–71. Springer Berlin / Heidelberg (2014)
28. Vicente, F.F.R., Lopes, F.M., Hashimoto, R.F., Cesar-Jr, R.M.: Assessing the gain of biological data integration in gene networks inference. BMC Genomics 13(Suppl 6), S7 (2012)
29. Wang, Z., Gerstein, M., Snyder, M.: RNA-Seq: a revolutionary tool for transcriptomics. Nat Rev Genet 10(1), 57–63 (2009)
30. Yilmaz, A., Mejia-Guerra, M.K., Kurz, K., Liang, X., Welch, L., Grotewold, E.: Agris: the arabidopsis gene regulatory information server, an update. Nucleic Acids Research 39(suppl 1), D1118–D1122 (2011)