IME - Instituto de Matemática e Estatística

Universidade de São Paulo
Instituto de Matemática e Estatística

USP - Universidade de São Paulo

Adequabilidade do Kepler para pacotes de Bioinformática do R

ALUNOS:

André de Melo Folli

Bruno Henrique Yoshimura

Gustavo Andrade Nery

ORIENTADOR

Prof. Dr. Roberto Hirata Jr.

2/19/2008

Índice

1. Instrução 5
2. Bioinformática 6
   2.1. Análise de Microarray 6
   2.2. O DNA Microarray 6
   2.3. Processo de obtenção de lâminas microarray 7
   2.4. Método 1 - Análise de Microarray de cDNA 7
      2.4.1. Stanford para obtencão de lâminas microarray (cDNA) 7
      2.4.2. Processo de Hibridização 8
   2.5. Método 2 - Análise de microarray comercial e Affymetrix (*nucleotídeos*) 9
   2.6. Detalhes sobre e*scaneamento* e reconhecimento de imagens 11
      2.6.1. Primeiro passo - Gradeamento 12
      2.6.2. Segundo paso - Segmentação 14
      2.6.3. Terceiro Passo – Quantificação (Intensidade do Spot) 17
3. Necessidade da computação para a Biologia 20
4. R Project 21
5. CRAN 23
6. Projeto Bioconductor 23
7. Programação Visual (*VP*) 25
   7.1. Introdução 25
   7.2. Definição 25
   7.3. Linguagens de Programação Visual (LPV) 26
   7.4. Benefícios, Vantagens e Estratégia 26
   7.1. Críticas, Problemas e Desafios 30
      7.1.1. Domínio Limitado 30
      7.1.2. Resistência da comunidade 30
      7.1.3. Escalabilidade 30
      7.1.4. Poucas ferramentas e falta de eficiência 31
      7.1.5. Abstração Procedural 32
      7.1.6. Abstração e Encapsulamento de Dados 32
      7.1.7. Checagem de tipos de dados 32
      7.1.8. Persistência 32
      7.1.9. Ferramentas além-programação 32
      7.1.10. Transparência da interface 32
      7.1.11. Práticas de Programação 33
8. Estudo de Caso 34
   8.1. Maiges Pack 34

8.1.1. Introdução 34
8.1.2. Estrutura do pacote 34
8.2. Funcionamento do pacote 35
8.3. Execução do Tutorial 37
9. Kepler 40
9.1. O que é o Kepler? 40
9.2. Funcionamento do Kepler 40
10. Workflows Científicos 42
11. Avaliação sobre a adequabilidade 43
12. Conclusão 44
13. Bibliografia 46

Índice de Figuras

Figura 1: Diagrama que mostra processo de transcrição do DNA em RNA. 6
Figura 2: Imagem de hibridização mostrando spots verdes, vermelhos e amarelos. 7
Figura 3: ArrayMaker da Stanford University (“Home-made”) 8
Figura 4: Parte do resultado da digitalização de uma lâmina de microarray. 9
Figura 5: Imagem que ilustra o processo de fotoligrafia 10
Figura 6: Imagem que mostra o resultado final da produção do GeneChip 10
Figura 7: Imagem retirada do site [35] Genepix Pro 11
Figura 8: Algoritmo “deformable gridding” proposto por Lipori. 12
Figura 9: Rotação de -20o para corrigir imagem. 13
Figura 10: Esta figura mostra como uma correção proposta por Battiato. 14
Figura 11: A linha amarela representa o resultado do Seeded Region Growth. 15
Figura 12: Histograma de um Spot 16
Figura 13: Regiões de “background” consideradas pelos softwares comerciais 18
Figura 14: Exemplo de programa que simula o movimento de um robô 28
Figura 15: Construção de um ciclo utilizando a LPV da Microsoft Robotics 33
Figura 16: Diagrama de classes do pacote MAIGES 35
Figura 17: Execução do tutorial do pacote maiges no R 39
Figura 18: Implementação no Kepler do pacote Maiges 41
Figura 19: Exemplo de workflow científico que recebe dados científicos 42

1. Instrução

Na medida que o estudos na área de Biologia Molecular avaçam, surge a necessidade de novas ferramentas de análise. Esse contexto requer a interação entre pesquisadores da área de biologia com estudiosos da área de computação, a fim de desenvolver essas ferramentas em conjunto.

Uma nova e importante área de biologia que interage com a computação é a *análise de microarray*. Uma tecnologia capaz de medir a expressão de milhares genes simultaneamente. Esta área envolve a computação em diversas etapas: escaneamento e análise de lâminas de *microarray*, normalização de dados, análise exploratória e estatística.

Uma característica deste tipo de análise que é importante para o nosso estudo, é a sua heterogeneidade. Não existe uma única sequência de passos a serem seguidos em uma análise para todos os tipos de dados. Análises podem utilizar diferentes métodos estatísticos que devem ser escolhido antes do processo.

Isso abre a oportunidade para utilizarmos um sistemas de programação visual que facilitariam o processo e ofereceriam uma interface mais amigável para cientistas.

Iremos apresentar um estudo sobre a adequabilidade do software Kepler para integração com o R na execução do pacote de análise de *microarray* Maiges. Esse estudo está separado em duas partes, sendo a primeira um embasamento teórico necessário para o entendimento da segunda parte - um breve estudo de caso que complementa e justifica a parte teórica.

A primeira parte se inicia mostrando toda a complexidade para se obter dados de lâminas de *microarray* em diversos tipos de processo. Este passo envolve o escaneamento, reconhecimento de imagem, correção automatizada de problemas de leitura. Neste ponto fica clara a necessidade de um tratamento computacional.

Do mesmo modo, ilustramos a dificuldade encontrada pelos usuários na utilização dessas ferramentas computacionais. A partir disso, exibimos o conceito de programação visual e exploramos as facilidades e dificuldades que ela pode nos trazer.

Já a segunda parte apresentamos o pacote do R próprio para realizar análises de dados obtidos em *microarray* (Pacote Maiges) e mostramos como seria a execução dos primeiros passos de uma análise utilizando esse pacote. Fazemos isso com o intuito de exemplificar para o leitor as dificuldades encontradas pelos usuários do R.

Como solução, apresentamos uma ferramenta de programação visual (Kepler) que pode ser integrada ao R para tornar a sua execução mais simples para os usuários. Mostramos como poderia ser feita essa integração e terminamos com os problemas e facilidades encontrados com o Kepler.

2. Bioinformática

2.1. Análise de Microarray

Todos os seres vivos guardam em uma molécula de DNA (*ácido desoxirribonucleico*) informações vitais para o funcionamento do organismo. O processo de *expressão gênica* é responsável por transcrever pequenas porções do DNA (genes) nas moléculas de RNA.

Figura 1: Diagrama que mostra processo de transcrição do DNA em RNA.

As células e tecidos dos corpos funcionam corretamente quando os genes são expressos de forma correta. Qualquer expressão alterada ou realizada em momento incorreto, pode ocasionar patologias. Isso ocorre pois os genes interagem entre si de maneira complexa formando uma espécie de rede, e qualquer alteração em único gene pode encadear um processo patológico. O câncer, por exemplo, acontece quando células do corpo se multiplicam de forma descontrolada e é causada pela mutação na expressão dos genes responsáveis pela divisão celular.

Entre os principais estudos da Biologia Molecular, está a análise da expressão gênica em diversos tecidos biológicos a fim de entender e diagnosticar tecidos patológicos. A análise de *microarray* é uma tecnologia capaz de medir a expressão gênica de milhares de genes simultaneamente. Este é um processo muito complexo, pois não existe um procedimento genérico que possa ser amplamente utilizado em uma análise *microarray*.

2.2. O DNA Microarray

DNA microarray é um arranjo de moléculas DNA ligadas a lâminas (*chips*) com pontos (*spots*) marcados sobre elas. Cada spot, pode conter milhões de moléculas DNA idênticas e cada *microarray* pode conter centenas ou milhares de *spots*. Cada spot, por sua vez, pode representar um gene diferente.

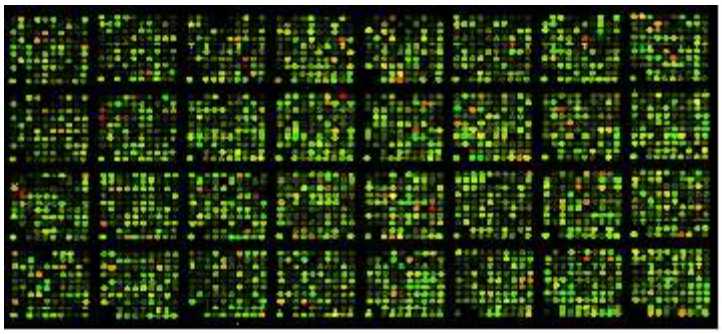

Figura 2: Imagem de hibridização mostrando spots verdes, vermelhos e amarelos. Figura retirada de [9] , "Métodos estatísticos para a análise de dados de cDNA microarray em um ambiente computacional integrado"

A idéia da análise de microarray é utilizar uma amostra com muitos DNAs idênticos e quantificar a capacidade dos mRNAs de se hibridarem com o DNA da amostra. A detecção é possível pela marcação com fluorocromos verde e vermelho. Desta forma, a imagem de hibridização da lâmina é obtida e pode ser *scaneada*.

2.3. Processo de obtenção de lâminas microarray

Vamos apresentar dois métodos de preparação das lâminas de *microarray*. O primeiro deles (*Stanford [19]*) utiliza **sequências de cDNA (método 1)**, que podem ser obtidas a partir de amostras de mRNA usando o processo de transcrição reversa. Este processo utiliza amplificação por PCR para fixação nos *spots*. O segundo é um método comercial, conhecido como *Affymetrix [25]* e utiliza **oligonucleotídeos (método 2)** produzidos na própria lâmina.

A seguir, mostraremos os dois tipos de análise de microarray, desde a preparação das lâminas.

2.4. Método 1 - Análise de Microarray de cDNA

O diagrama abaixo ilustra uma rotina básica de análise de microarray, desde a obtenção e amostras até a apresentação dos resultados.

> Amostras → Extração do RNA → Controle de qualidade da amostra → Marcação → Hibridização da lâmina → Scanner → Tratamento e análise da imagem → Normalização dos dados → Análise do dados → Conclusão

Abaixo apresentaremos o processo de obtenção e hibridização de lâminas cDNA.

2.4.1. Stanford para obtencão de lâminas microarray (cDNA)

Em 1995, *Dr. Patrick Brown* e *Ron Davis* da *Stanford University* publicaram este método [19] de preparação de lâminas *microarray* que é realizada por impressão de fragmentos inteiros de DNAs obtidos a partir do mRNA. A obtenção dessas lâminas é feita por um robô, responsável pela impressão nas lâminas de vidro. Após a impressão, as moléculas de mRNA são transcritas em cDNA para torná-las mais

estáveis. Finalmente, as moléculas de cDNA podem ser marcadas com corantes fluorescentes como veremos no capítulo a sobre hibridização.

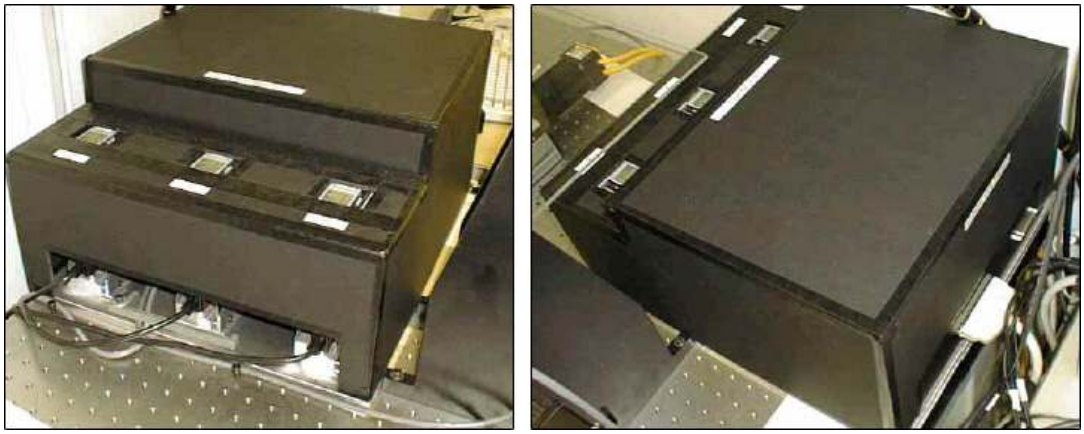

Figura 3: ArrayMaker da Stanford University ("Home-made")

A grande vantagem na utilização deste método é seu baixo custo. Por este motivo, o Método Stanford também é chamado de "*home-made*". No sitio do *The Brown Lab*, da *Stanford University*, existe um guia completo para construção de um robô (http://cmgm.stanford.edu/pbrown/).

2.4.2. Processo de Hibridização

Antes do processo de hibridização, precisamos extrair o mRNA (RNA mensageiro) de nossa amostra experimental. Assim, o mRNA poderá hibridizar o cDNA por transcrição reversa. São utilizadas duas amostras diferentes de cDNA: uma delas é marcada com Cy3 (verde) e outra com Cy5 (vermelho). Em cada um dos *spots* da lâmina, correspondente a um gene, conseguimos obter as proporções de mRNAs das duas amostras.

Caso a fluorecência do spot seja semelhante para Cy3 e Cy5, a expressão gênica pode ser considerada inalterada. Fluorescência avermelhada (Cy5) indica amostra inativa, enquanto que fluorescência esverdeada (Cy3) indica amostra ativa.

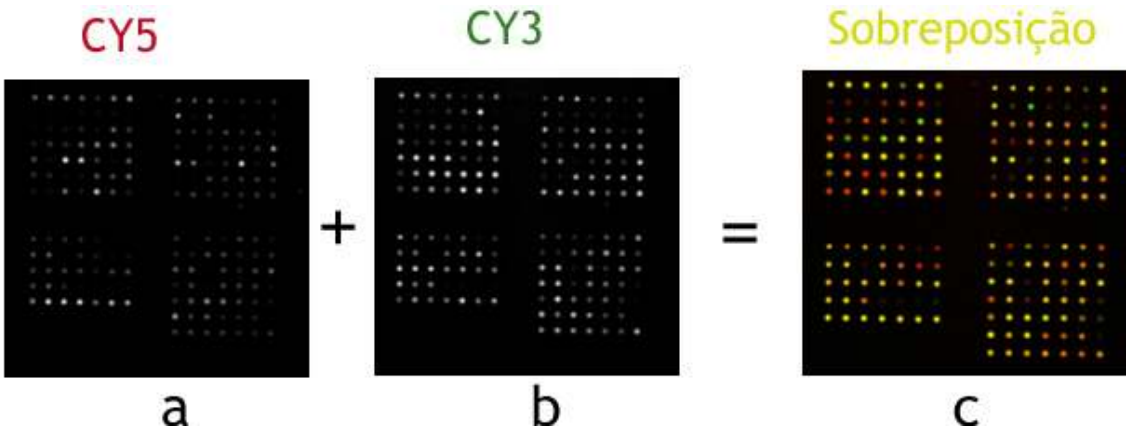

Figura 4: Parte do resultado da digitalização de uma lâmina de microarray. O canal Cy5 (a), o canal Cy3 (b) e a composição das duas imagens (c). Imagem retirada de [36] Microarray Image Analysis

A lâmina pode finalmente ser scaneada para que a imagem de hibridização seja processada por algum programa de reconhecimendo de imagens.

2.5. Método 2 - Análise de microarray comercial e Affymetrix (*nucleotídeos*)

Affymetrix [23] é uma solução comercial para obtenção e análise de lâminas *microarray* de oligonucletídeos. Essas lâminas são chamadas comercialmente de *GeneChip®* e são obtidas pelo processo de fotoligrafia.

A produção dos microarrays *GeneChip® [25]* utiliza o conhecimento da indústrias de semicondutores neste processo, que é baseado na utilização de luzes (radiação ultravioleta) para produção de múltiplas camadas de certo material (no nosso caso, camadas de nucleotídeos para formar o oligonucleotídeo).

A produção da lâmina de oligonucleotídeos ocorre da seguinte forma: os nucleotídeos (Adenina, Guanina, Timina, e Citosina) são ligados em uma longa cadeia sequencial formando um *oligonucleotídeo* (na ordem correta). Utilizando uma máscara e luz ultravioleta (fotoligrafia), é possível direcionar os nucleotídeos para uma parte específica da lâmina (*spot*), estimulando a ligação com o pedaço já produzido dessa cadeia. Máscara após máscara, a sequência vai se formando até resultar nos oligonucleotídeos desejados. Veja na imagem abaixo as janelas das máscaras que permitem a entrada de luz e resultam na adição de nucleotídeos na cadeia.

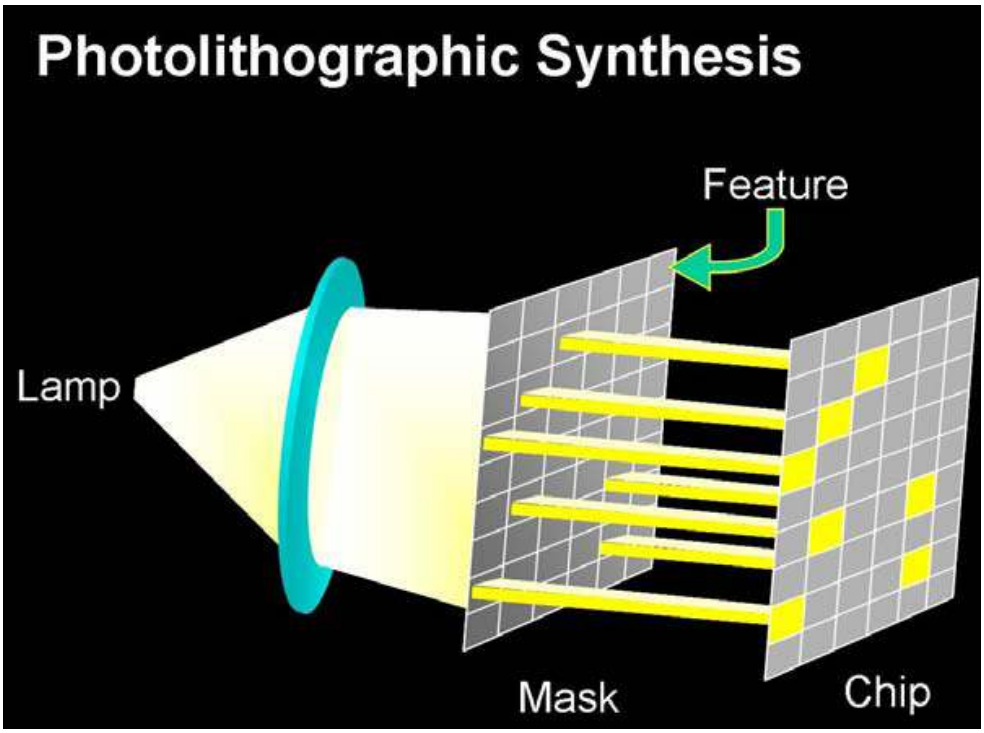

Figura 5: Imagem que ilustra o processo de fotoligrafia e mostra como a máscara pode filtrar a luz ultravioleta. (Imagem redirada de [25] "GeneChip® Microarrays - Activity #3 – Manufacturing of GeneChip Microarrays and Building Models")

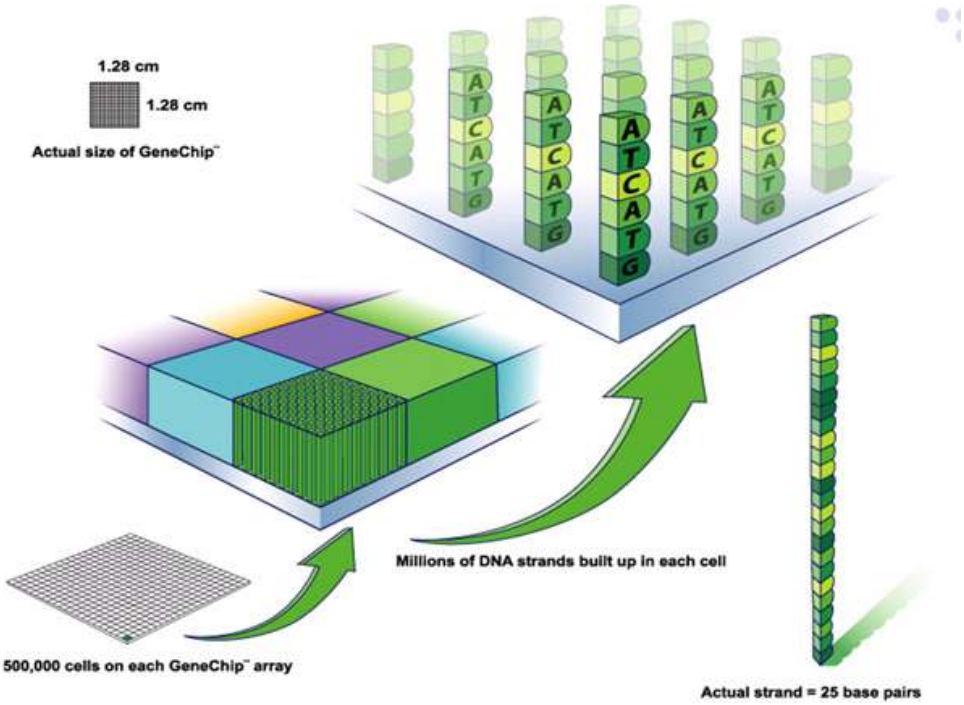

Figura 6: Imagem que mostra o resultado final da produção do GeneChip, já com os seus oligonucleotídeos formados na lâmina. (Retirada de [25] "GeneChip® Microarrays - Activity #3 – Manufacturing of GeneChip Microarrays and Building Models").

Enfim, a lâmina está pronta para a hibridização e coloração (feita pelo *GeneChip® Hybridization Oven*). Agora as lâminas podem ser escaneadas (pelo *GeneChip® Scanner*) e tratadas por um programa (*GeneChip® Operating Software*) que irá quantificar e analisar a intensidade da fluorescência em cada *spot*.

2.6. Detalhes sobre e*scaneamento* e reconhecimento de imagens

A análise de dados só é possível após o *escaneamento* e varredura da imagem obtida a partir das lâminas. Para isso, existem os programas que utilizam algoritmos de reconhecimento de imagem. Alguns exemplos de programas são *GenePix Pro, QuantArray, TIGR Spotfinder, ArrayVision, GridGrinder*, *GeneChip® Operating Software* entre outros.

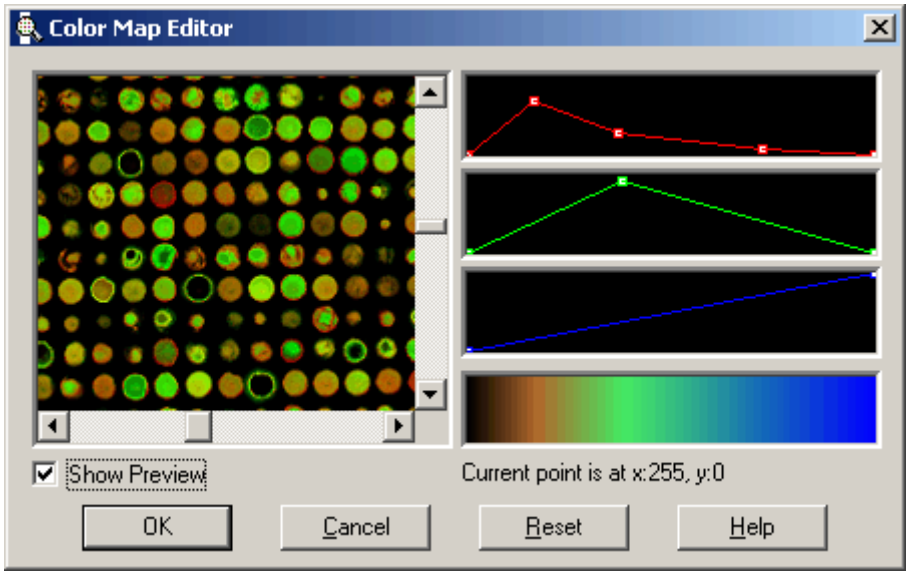

Figura 7: Imagem retirada do site [35] Genepix Pro

Podemos dividir em dois tipos os escaners disponíveis no mercado:

1. *Laser* – Digitaliza a imagem baseada na diferença do comprimento da fluorescência vermelha e verde. Neste caso é gerada uma imagem para o comprimento de onda cy3 e outra para cy5 (comprimento de 570nm e 670nm, respectivamente).
2. *Charge Coupled Device (CCD)* – Utiliza luz branca e uma câmera fotográfica com lente objetiva, para garantir contraste dos tons.

As imagens obtidas com os escaners são 16-bit e o processo de análise e correção de imagem pode ser divido em três etapas principais:

1. *Gradeamento* – Divisão da imagem em grade, cujos pedaços mínimos representam *spots* com *background*.

2. *Segmentação* – Separação do *background* com o *foreground (*região sinal*),* para diferenciar a área de interesse da cor de fundo (*foreground* é o próprio *spot*). Isso não significa que descartaremos as informações do *background*.
3. *Quantificação* – O *foreground* extraido no segundo passo será analisado quanto a intensidade da fluorescência verde e vermelha (R e G no padrão RGB). Nesta etapa utilizaremos o *background* para correção de ruídos.

Abaixo, mostraremos com maiores detalhes cada etapa da análise e correção da imagem.

2.6.1. Primeiro passo - Gradeamento

O primeiro passo na análise da imagem da lâmina obtida no e*scaner* é o **gradeamento**. O objetivo desse processo é transformar a imagem em uma matriz (grade), determinando a região de cada spot que será analisada posteriormente na segmentação. Essas regiões são normalmente retangulares e contém o *foreground* (região de sinal) e o *background*.

Alguns softwares de gradeamento (*GenePix Pro, ScanAlyzer*) requerem alguns parâmetros para ajudar na leitura da imagem. Este tipo de análise é chamada de **baseada em template**, e os dados necessários são:

- número de colunas e linhas;
- número de grades;
- espaçamento entre colunas e linhas;
- altura e largura do spot ;
- translação da imagem (caso a lâmina não estivesse na posição correta);
- posição da primeiro spot em relação a lâmina.

Também existem os softwares que analisam a própria imagem em busca dos parâmetros acima, automatizando o processo de alinhamento e conseguindo encontrar os spots mesmo em grades não uniformes. São os chamados **softwares baseados em dados** (*Data-driven softwares*).

A acurácia do gradeamento pode ser aumentada permitindo interação humana. Porém, a eficiência será prejudicada. Por este motivo, existem softwares que permitem a escolha entre gradeamento automático ou manual.

Na prática, as imagens obtidas no *escaneamento* podem apresentar deformações, divergência de espaçamento entre spots, desalinhamento entre linhas e rotações. Um algoritmo proposto por *Lipori [30]* chamado de "*deformable gridding*" pode ser utilizado para posicionar os spots de forma que fiquem centralizados e uniformes na imagem de análise.

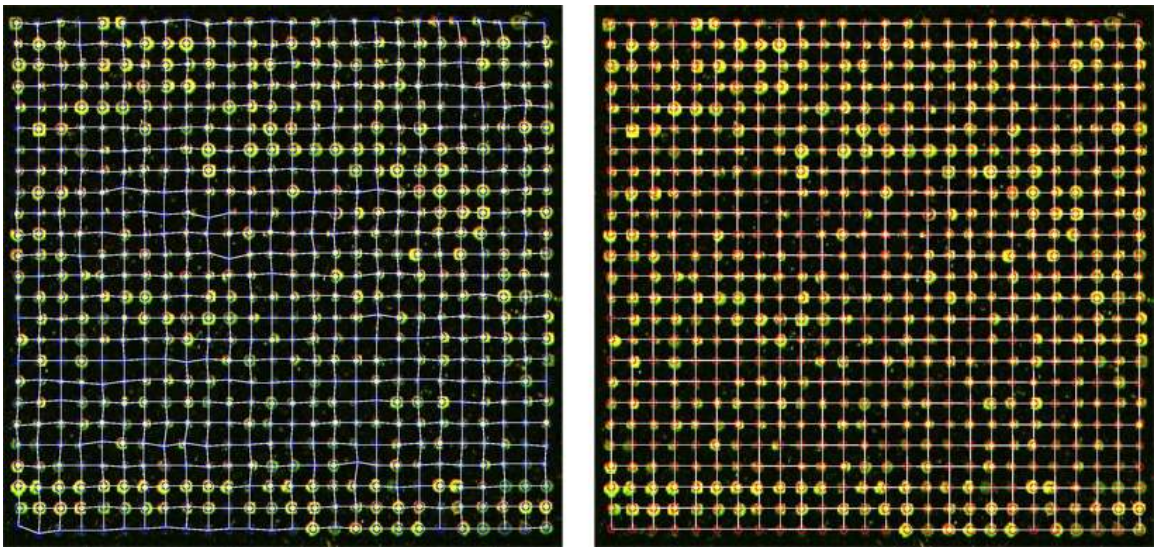

Figura 8: Algoritmo "deformable gridding" proposto por Lipori. Imagem de antes e depois em dados hipotéticos. (Retirada de [30] Efficient Gridding of Real Microarray Images", Giuseppe Lipori)

O método é baseado no Orientation Matching Transform (OMT) e pode ser usado tanto para os softwares que recebem os parâmetros de entrada acima descritos, quanto para os automatizados.

No caso da imagem obtida apresentar **rotação**, podemos repará-la facilmente com interação humana. Entretando, existe uma forma automatizada de fazê-la.

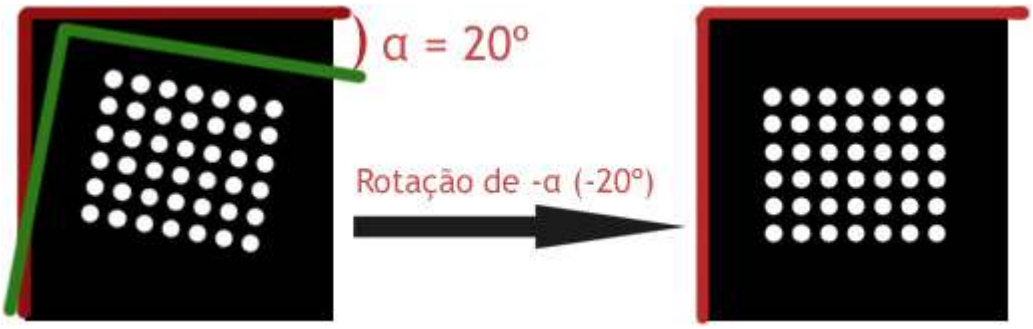

Figura 9: Rotação de -20o para corrigir imagem.

Uma maneira de resolver o problema de forma automatizada é fazer uma busca exaustiva de um ângulo α, dentro do intervalo $[\alpha_{min}, \alpha_{max}]$, que maximize a combinação dos valores da projeção RT (*Randon Transform*) na direção das colunas (vertical) e linhas (horizontal) da imagem rotacionada em α. Isto é, rotacionar a imagem de forma que maximize as projeções para encontrar o melhor ângulo de rotação *[Battiato et al., '07] [33]*.

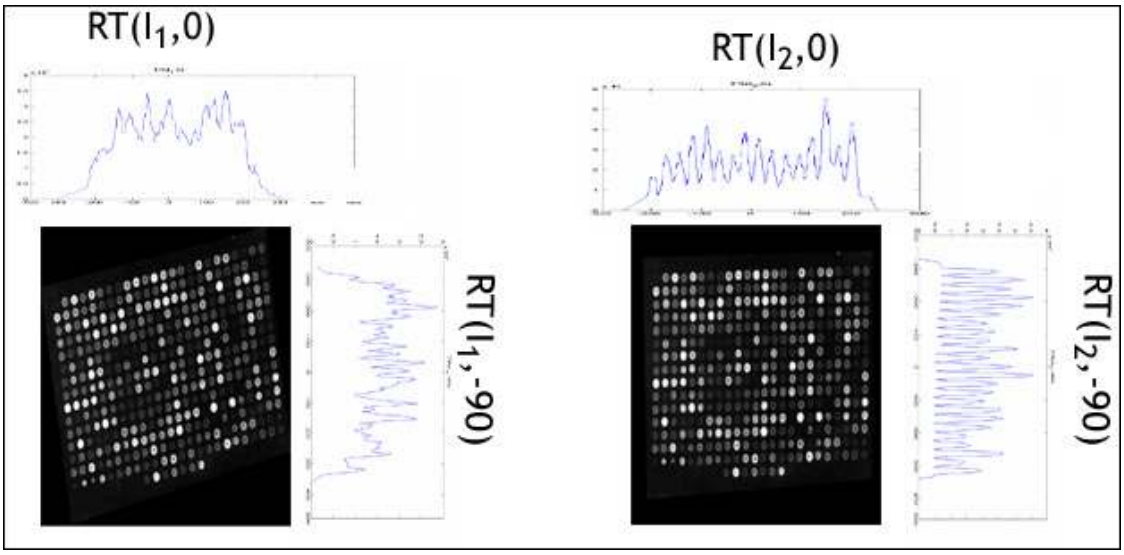

Figura 10: Esta figura mostra como uma correção proposta por Battiato funciona. Os gráficos de projeção da primeira figura não apresentam "picos" e "vales" bem deifnidos (intensidade alta a baixa, respectivamente). Já a segunda imagem estes estão muito bem definidos. Imagem retirada de [36] Microarray Image Analysis.

A figura acima ilustra os gráficos de projeção de intensidade I na direção α. Ou seja, o gráfico de RT(I, α). No caso, assumimos que a Imagem 2 (I_2) já estava rotacionada no melhor angulo.

2.6.2. Segundo paso - Segmentação

Como citamos anteriormente, este é o processo de separação do *background* com o *foreground (*região sinal*).* Apenas depois de determinada a região de sinal, poderemos enfim quantificar a intensidade das fluorescências.

Existem diversos métodos de segmentação e suas diferenças se baseiam em:

- diferentes formatos de *spots* que podem ser reconhecidos;
- eficácia *vs.* abrangência de formas
- análise de imagem *vs.* modelos estatísticos.

Abaixo vamos mostrar os métodos mais utilizados, a maioria deles descritos por *Yang em "Comparison of methods for image analysis on cDNA microarray data (2000)" [26].*

2.6.2.1.Fixed Circle Segmentation (FC)

Esta primeira forma de segmentação assume um tamanho fixo para o diâmetro de todos os *spots*. Como consequência, alguns *spots* serão lidos parcialmente e podem ser confundidos com o próprio *background*.

2.6.2.2.Adaptative Circle Segmentation (AC)

Este método estima o valor do diâmetro de cada spot separadamente, mas ainda sofre perdas na leitura pois não aceita deformações na forma do *spot*. Esta prática também consome bastante tempo de processamento pois precisa analisar imagem de milhares de *spots* separadamente. Este método é utilizado pelo *QuantArray*.

2.6.2.3.Adaptative Ellipse (AE)

Em 2005, *Luis Rueda* and *Li Qin [27]* propuseram uma nova forma de segmentação que corrige parte do problema das duas soluções acima. Utilizando o processo de diagonalização, ele é capaz de ler os *spots* como *elipses* ao invés de círculos. Isso significa uma maior abrangência de formas, pois um círculo é um caso particular de elipse.

Entretando, existem muitas outras formas de *spot*. Entre elas estão os formatos de roscas*, e*lipses deformadas, entre outros.

2.6.2.4.Adaptive Shape Segmentation (AS)

Um dos métodos utilizados nesta segmentação é o *Seeded Region Growth* (SRG). O processo de descoberta do *spot* começa em uma semente (*seed*) que precisa ser conhecida. Depois percorre os pixels vizinhos para determinar seu *foreground* e *background*.

A descoberta das sementes pode ser feita na fase de gradeamento e podemos utilizar o algoritmo proposto por *Lipori* para centralizar as sementes e simplificar a segmentação.

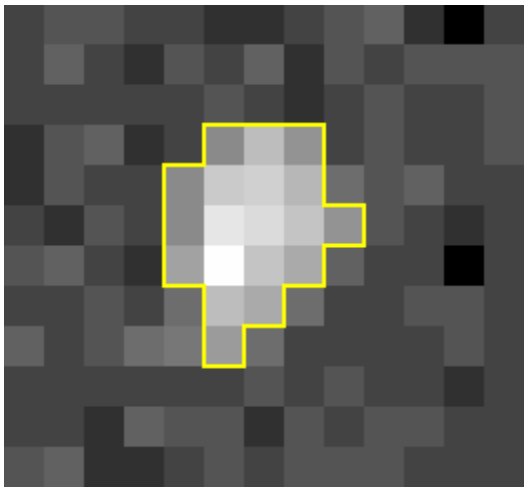

Figura 11: A linha amarela representa o resultado do Seeded Region Growth. [Yan et al., 2000].

2.6.2.5.Histogram Segmentation(Hist.)

Este método utiliza o histograma de intensidade do *foreground* e *background* e não fica restrito a nenhuma forma específica (círculo, elipse, elipses deformadas). Antes de analisar o histograma, utiliza-se uma máscara que pode ter formato elíptico ou retangular. As janelas desta máscara precisam ser suficientemente maiores do que todos os *spots* (na prática podemos ter janelas maiores do quase todos os *spots*).

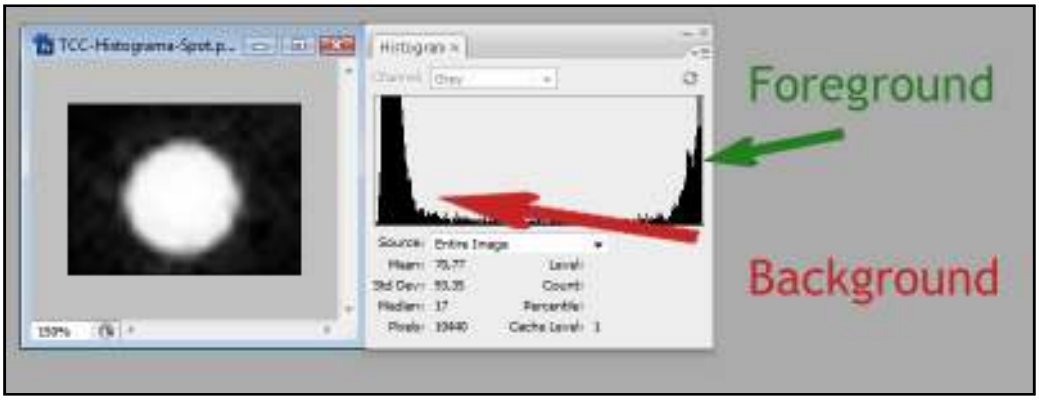

Figura 12: Histograma de um Spot mostrando a diferença entre background e foreground.

Apesar de sua simplicidade, este método pode ser pouco eficaz caso o tamanho da janela seja muito grande. Uma das formas de contornar este problema foi implementada no *QuantArray*: a área da janela escolhida é baseada no percentual entre a área total do *foreground* e *background*, sendo que o primeiro varia de 5% a 20%, enquanto que o segundo de 95% a 80%.

2.6.2.6.Mann-Whitney segmentation

Este algoritmo de segmentação (*Chen et al.*, 1997)[37] é baseado no teste estatístico *Mann-Whitney* e calcula o limiar (*threshold*) entre o *foreground* e *background*. O primeiro passo é criar uma máscara com o provável *foreground*. Em seguida, escolhe pixels aleatoriamente no *background* e comparar com uma quantidade de pixels (pré-determinada) com baixa intensidade do *foreground* (com o teste *Mann-Whitney*). Se a diferença entre as distribuições for insignificante, pixels do foreground são descartados. Os pixels que sobrarem no *foreground* são o próprio *spot*.

2.6.2.7.K-means segmentation

Este tipo de segmentação se baseia na clusterização *K-Means (Bozinov and Rahnenführer, 2002)* [36] , que envolve o problema dos quadrados mínimos. Nesse caso específico, k é igual a 2 e as duas regiões estudadas são *background* e *foreground*.

2.6.2.8.Tabela Comparativa das formas de segmentação

Esta tabela abaixo mostra formatos de *spot* e a capacidade de leitura dos mesmos pelos diferentes tipos de segmentação. Receber a avaliação negativa indica que esta forma de segmentação não é perfeita para leitura de tal formato, mas não significa que não será lida de forma correta.

Fixed Circle	*Não*	*Não*	*Não*	*Não*	*Não*
Adaptative Circle	*Sim*	*Não*	*Não*	*Não*	*Não*
Adaptaicve Ellipse	*Sim*	*Sim*	*Não*	*Não*	*Não*
Adaptative Shape	*Sim*	*Sim*	*Sim*	*Sim*	*Sim*
Histogram	*Sim*	*Sim*	*Sim*	*Sim*	*Sim*

Tabela 01 – Comparação entre softwares quanto ao reconhecimento de formas.

2.6.3. Terceiro Passo – Quantificação (Intensidade do Spot)

Para chegarmos a quantificação final dos *spots* (ratio) , precisamos analisar localmente as intensidades do *foreground* e *background* para cada fluorescência. Em seguida, tirar o ratio da fluorescência para Cy3 e Cy5 .

2.6.3.1.Intensidade do Foreground

A etapa final do processo de análise de imagem é extrair a intensidade de cada *spot*. Sabemos que a quantificação para a "taxa de hibridização" de um *spot* é diretamente proporcional a fluorescência do mesmo. Podemos calcular a **intensidade do *foreground*** com:

- média das intensidades do *spot;*
- mediana das intensidades do spot;

Para conseguir maior consistência na análise, os softwares também extraem a intensidade do *background* a fim de minimizar os efeitos causados por ruídos (reação do vidro com componentes químicos, umidade da lâmina, sujeira na câmera de leitura).

2.6.3.2.Intensidade do Background

Entre os métodos mais utilizados para medir a **intensidade do *background*** estão:

- a média das intensidades dos pixels ao redor do *spot*;
- mediana dos pixels ao redor do *spot*;
- constante global para todos *spots*;
- nenhum método.

Veja na imagem abaixo as regiões que os softwares consideradam como *background* [*Yang et al.,* 2001].

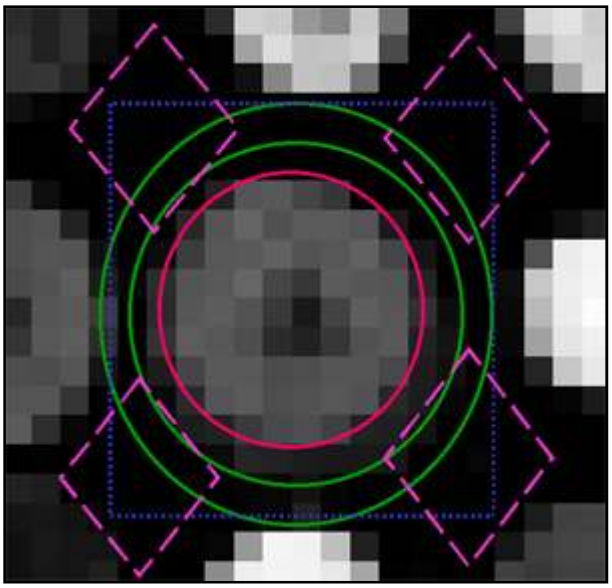

Figura 13: Regiões de "background" consideradas pelos softwares comerciais. A região em vermelha representa o spot; a região compreendida entre os círculos verdes é o background considerado pelo QuantArray;Regiões de "background" consideradas pelos softwares comerciais. A região em vermelha representa o spot; a região compreendida entre os círculos verdes é o background considerado pelo QuantArray; a região interna do quadrado azul e externa ao círculo vermelho é a região considerada pelo Scanalyze, enquanto que os quadriláteros rosas são utilizados no software Spot.

2.6.3.3.Cálculo do Ratio

Finalmente, após escolher as opções de medição de intensidade de *background* e *foreground*, podemos quantificar nosso experimento. Vamos utilizar a média da intensidade para exemplificar.

Intensidade Vermelha = media(Red Foreground) – media(Red Background)

Intensidade Verde = media(Green Foreground) – media(Green Background)

E, por fim

$$\text{Ratio} = \text{Log}_2 (\text{Intensidade Vermelha}/\text{Intensidade Verde})$$

Podemos concluir que:

- **ratio > 0** significa *spot* com **maior** intensidade;
- **ratio < 0** significa *spot* com **menor** intensidade.

A notação logarítmica na base 2 pode ser utilizada para aproximar nossos resultados para uma Distribuição Normal. Caso não utilizemos esta transformação, o ratio poderia ser analizado da seguinte maneira:

Ratio = (Intensidade Vermelha/Intensidade Verde)

- ratio variando no intervalo (1,+ ∞) significa *spot* com **menor** intensidade;
- ratio variando entre (0,1) significa *spot* com **maior** intensidade.

3. Necessidade da computação para a Biologia

Como visto anteriormente, o processo de obtenção de dados através da análise de *microarray* é bastante complicado e custoso. Além disso, esse processo gera uma quantidade muito grande de dados para serem tratados e, para fazer a análise desses dados, são utilizados diversos métodos matemáticos e estatísticos que podem ser complexos e ainda ter uma quantidade elevada de dados a serem analisados. Devido a essas dificuldades, se fez necessária a introdução de um ambiente computacional.

Infelizmente, a união da computação com a biologia não é tão simples quanto se possa imaginar. Para implementar os diversos métodos a serem aplicados nos dados, para que a entrada e saída de dados sejam feitas corretamente e para que todas as outras etapas do processo computacional envolvido funcionem como o esperado, é preciso da colaboração de pessoas da área de computação (principalmente no desenvolvimento dos *softwares* envolvidos) e da área de biologia (tanto na explicação de como os programas devem trabalhar quanto na validação dos mesmos). Essa colaboração nem sempre é simples. Este é um dos motivos que para o surgimento de estudiosos em Bioinformática, com conhecimentos tanto de biologia, como de computação.

Apesar de todas essas dificuldades enfrentadas no desenvolvimento, atualmente existem diversos softwares para análise de dados obtidos através de experimentos de microarray, agora vamos falar um pouco mais sobre um deles, o R.

4. R Project

O R é um software livre sob a licença da "*Free Software Foundation's GNU General Public License*", de código aberto e multi-plataforma. Devido à facilidade de adquirir, instalar e rodar o R em diversas plataformas, existe grande colaboração entre seus usuários, que trazem melhoramentos e novas funcionalidades.

O R é uma linguagem de programação e também um ambiente para computação estatística e produção de gráficos. Ele provê uma grande variedade de técnicas estatísticas (modelagem linear e não linear, testes estatísticos clássicos, análises de séries temporais, classificação) e gráficas (facilidade para produzir gráficos de excelente qualidade, incluindo símbolos matemáticos e fórmulas), e é facilmente extensível. Isso ocorre pois o ambiente R, assim como o S, foi feito utilizando uma linguagem de programação completa, permitindo que usuários definam suas próprias funções.

Grande parte do sistema foi escrita em R, possibilitanto que seus usuários acompanhassem as escolhas algorítmicas feitas (uma vez que o código é aberto). Para tarefas computacionalmente intensivas, podem ser feitas rotinas nas linguagens de programação em C, C++ e Fortran. Estas, podem ser facilmente chamadas em tempo de execução. Usuários avançados podem escrever código C para manipular objetos R diretamente, e isto pode ser muito interessante na área de optimização de performance.

O usuário, além de poder escrever as suas próprias funções, tem a possibilidade de estender as funcionalidades do R via pacotes. Os pacotes do R são bibliotecas de funções úteis para determinada área de estudo. O ambiente vem com alguns pacotes básicos instalados e muitos outros estão disponíveis em sites específicos. O R possui sua própria documentação no formato LaTeX

A linguagem R pode ser considerada um dialeto, ou uma implementação diferente da linguagem S. Apesar de existirem algumas importantes diferenças entre as duas linguagens, há muito código escrito em S roda normalmente no R.

Inicialmente o R foi escrito por *Ross Ihaka* e *Robert Gentleman* no Departamento de Estatística da *Universidade de Auckland*, Nova Zelândia. Depois disso um extenso grupo de pessoas contribuiu com o R mandando código e relatórios de bugs encontrados. Desde 1997 o núcleo de pessoas que pode modificar arquivos fonte da versão oficial do R é composto dos seguintes membros: *Doug Bates*, *John Chambers* (que foi o criador da linguagem S junto com sua equipe na "*Lucent Technologies*"), *Peter Dalgaard, Robert Gentleman, Kurt Hornik, Stefano Iacus, Ross Ihaka, Friedrich Leisch, Thomas Lumley, Martin Maechler, Duncan Murdoch, Paul Murrell, Martyn Plummer, Brian Ripley, Duncan Temple Lang, Luke Tierney* e *Simon Urbanek.*

Como dito anteriormente, o R pode ter suas funções estendidas quando instalamos os pacotes, esses pacotes ficam armazenados em repositórios na internet.

Nos próximos parágrafos falaremos um pouco sobre dois importantes repositórios para o nosso trabalho, o CRAN e o Bioconductor.

5. CRAN

O CRAN (*Comprehensive R Archive Network*) é um site que reúne uma rede de servidores FTP e WEB ao redor do mundo onde estão hospedados os códigos-fonte e binários do R.

Nele também estão armazenados uma infinidade de pacotes do R para diversas áreas da estatística, produção de gráficos e ainda toda a documentação referente. No site cada pacote possui uma breve descrição, versão, autoria. Ao fazer o download de um pacote, o usuário pode instalá-lo, ler sua documentação, obter exemplos de uso e tutoriais.

O CRAN é o principal repositório de pacotes do R, porém, para trabalhos em áreas mais específicas, como a Bioinformática, existem alguns sites com pacotes voltados a esse tipo de necessidade, como o Bioconductor.

6. Projeto Bioconductor

Bioconductor é um projeto de software de código aberto e de livre colaboração (interessados podem ajudar em seu desenvolvimento) que tem como objetivo prover um software robusto, com métodos estatísticos e matemáticos para a análise e compreensão de dados genômicos, boa documentação e facilidade na reprodução dos resultados obtidos.

Ele é baseado primariamente na linguagem de programação R, porém são aceitas contribições em qualquer outra linguagem de programação. O projeto começou no outono de 2001.

O núcleo de desenvolvimento do projeto é baseado primariamente no pessoal do Centro Fred Hutchinson de Pesquisa em Câncer. Existem também outros membros em instituições ao redor do mundo.

Os esforços iniciais do projeto Bioconductor eram focados principalmente na análise de dados obtidos a partir de análise de microarray especificamente, porém muitas das ferramentas desenvolvidas são bastante abrangentes e podem ser amplamente usadas para análises de diversos dados genomicos.

No site do projeto Bioconductor, assim como no CRAN , existe um repositório que armazena pacotes próprios para serem instalados no R. Nesse caso os pacotes do projeto não são úteis para todo tipo de análise estatística, como dito anteriormente, eles têm como objetivo a análise e compreensão de dados genômicos exclusivamente. Os pacotes que ficam armazenados no Bioconductor, assim como os do CRAN, possuem toda a documentação, exemplos e etc.

Agora temos uma boa noção de algumas das ferramentas que estão disponíveis atualmente para fazer análises de dados obtidos em microarray. Apesar do que possa

parecer, o uso dessas ferramentas não é tão simples quanto parece, mais adiante há um estudo de caso que explicita isso e mostra que poderíamos ter maior usabilidade unindo essas ferramentas a um software de programação visual. Agora apresentamos algumas noções teóricas sobre programação visual que consideramos importante.

7. Programação Visual (*VP*)

7.1. Introdução

As linguagens de programação convencionais são difíceis de aprender e requerem habilidades e conhecimentos que muitas pessoas não possuem. O número de computadores pessoais cresce a cada dia e com eles a utilização de diversos programas. Entretanto, os usuários são incapazes de modificá-los para atender suas necessidades específica. Para permitir que o usuário final modifique e reconfigure o sistema, este deve prover diversas opções e flexibilidade, deixando-o cada vez mais complexo.

Neste ambiente que surge a idéia da Programação Visual (PV): a utilização de componentes gráficos e expressões visuais como linguagens, com a finalidade de tornar as tarefas de programação mais fáceis e conseqüentemente mais acessíveis aos usuários. Deste modo, será possível construir sistemas sem os pré-requisitos e conhecimentos avançados de programação, seja por programadores ou não. Os principais objetivos são: tornar a programação mais fácil de aprender, realizar tarefas complexas com menos erros e economia de tempo de desenvolvimento.

7.2. Definição

Programação Visual é qualquer sistema no qual o usuário escreve um programa em duas ou mais dimensões, uma vez que as linguagens textuais convencionais são interpretadas e compiladas como um fluxo longo e unidimensional (1).

De outra maneira, PV refere-se a qualquer sistema que permite a construção de programas manipulando elementos gráficos e metáforas visuais, e não apenas especificando-os textualmente como na programação convencional.

É importante explicitarmos a diferença entre Programação Visual e a Visualização Gráfica do Programa (VGP), pois são conceitos totalmente diferentes. Na PV, os elementos gráficos são usados para a construção do programa em si, diferentemente da VGP, na qual os elementos podem ser especificado tanto textualmente como da maneira convencional. Neste caso, os elementos gráficos são usados apenas para ilustrá-lo, seja em um aspecto específico ou em sua própria execução.

A VP se divide em visualização dos dados, do código e do algoritmo (execução do programa). Ainda podem ser classificadas quanto ao tipo de visualização: dinâmica ou estática. A primeira refere-se a animações do programa rodando e a segunda representa “fotos” do programa em determinado momento da execução.

Apesar de distintos, essses conceitos se unem de tal forma que a demonstração e depuração de um programa construído com PV deve ser feita de maneira gráfica, utilizando assim a VG. Deste modo, conseguimos separar mais precisamente os conceitos de construção do programa (por VP), e sua visualização gráfica após ser construído (por VGP).

7.3. Linguagens de Programação Visual (LPV)

Na literatura, encontramos diversas definições de linguagens de programação visual:

- Uma linguagem de programação visual é um conjunto de arranjos espaciais e símbolos gráficos com uma interpretação semântica quem se comunicam usados para representar as ações em um programa (1).

- São linguagens que permitem construir programas manipulando informações e expressões visuais num processo interativo (2).

De modo mais sucinto, são linguagens que usam representação visual no processo de programação, criando ou modificando um programa.

Essas expressões visuais (como gráficos, imagens, ícones, arcos, flechas, caixas e círculos), se conectam montando o fluxo de execução do programa. As linguagens de PV envolvem técnicas de interação com manipulação direta do usuário com os objetos. Por exemplo: selecionar (*"point-and-click"*) e mover (*"drap-and-drop"*). Isto torna a programação uma experiência mais agradável e intuitiva.

Vale ressaltar que Visual Basic, Visual C++ e toda família Microsoft Visual ™ não são linguagens de programação visual, mas sim linguagens textuais que apenas utilizam uma interface visual para que os programadores construam interfaces gráficas de forma mais rápida e fácil.

7.4. Benefícios, Vantagens e Estratégia

São inúmeras as vantagens do uso de gráficos e expressões visuais em ambientes computacionais. Dentre as várias razões, podemos citar como principal argumento o fato de que os humanos processam imagens e informações visuais de forma mais clara e otimizada.

Imagens são mais poderosas do que palavras como meios de comunicação, pois elas agregam mais informação e significado em uma unidade de expressão mais concisa. Também não possuem as barreiras que as linguagens impõe. Além disso, as pessoas lidam muito melhor com exemplos concretos e visuais do que idéias abstratas, pois são mais fáceis de aprender e lembrar.

Inclinando-se para o lado computacional, as linguagens de programação visual promovem um nível maior de abstração, fazendo-a de forma multidimensionale dando menos ênfase à sintaxe. Deste modo, ela passa a auxiliar a programação, tornando-a mais fácil de entender e de se reproduzir.

Podemos citar como exemplo os programas escritos em linguagens textuais. Se apresentados de forma visual com a identação de blocos ficam mais fáceis de serem lidos e compreendidos por outras pessoas.

```
void contaAteDez(){ int x;for(x = 1; x <= 10; x++){printf("%d",x);} }

void contaAteDez() {
    int x;

    for(x = 1; x <= 10; x++) {
        printf("%d \n", x);
    }
}
```

Exemplo de como uma melhor apresentação visual deixa o programa mais claro de ser lido, mesmo quando escrito utilizando-se linguagens textuais.

A utilização de gráficos para a PV faz com que a representação das informações e a manipulação dos dados sejam feitas de forma mais natural pelo usuário, aproximando-se de sua capacidade mental em realizar atividades do mundo real. Assim, ao desenvolver um programa de forma unidimensional (linguagens textuais comum), ele não desperdiça capacidade cerebral.

A PV tem o desafio de tornar a programação mais fácil de ser compreendida, seja por programadores experientes, novatos ou até mesmo não-programadores. A multidimensionalidade é diferença essencial entre linguagens de programação visual e as linguagens textuais.

Na área de desenvolvimento e manutenção, podemos citar outros benefícios da PV. A depuração de programas em sistemas visuais, por exemplo, agrega uma quantidade maior de informações do que em texto puro. A construção dos programas estimula a manipulação direta dos objetos e da interface, dando a sensação de maior poder ao programador.

Além disso, ela auxilia os programadores em alguns paradigmas normalmente difíceis de serem superados em linguagens convencionais (como é o caso da reusabilidade de componentes e a explicitação de relacionamentos).

A reusabilidade provê o benefício do desenvolvimento de métodos e na ajuda em construir novos programas que utilizam os componentes já desenvolvidos. Possibilita assim uma maior reusabilidade, portabilidade e interoperabilidade entre *softwares*. Já a explicitação de relacionamentos é muito útil, pois nos dá um retorno visual imediato das dependências dos objetos.

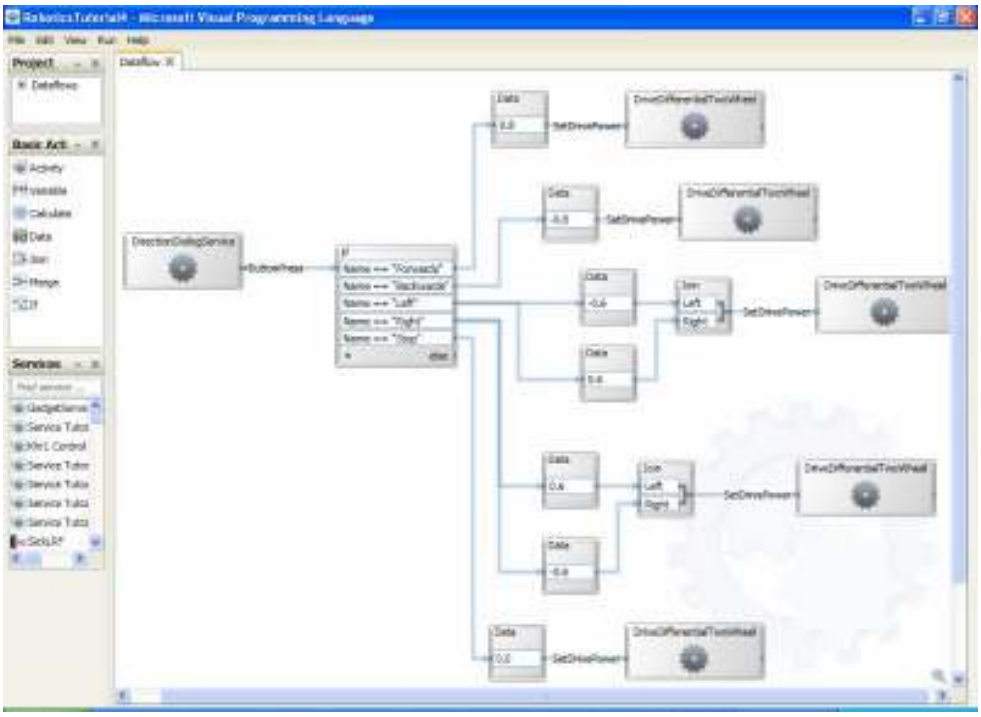

Figura 14: Exemplo de programa que simula o movimento de um robô [38]. Mostra a reusabilidade de componentes (DriveDifferentialTwoWheel). Microsoft Robotics.

Desta forma, a pesquisa em PV possui forte relação com outras áreas, como por exemplo a de Interação Homem-Computador (IHC, ou HCI em inglês. (3)). Esta área estuda a forma com a qual os usuários interagem com o sistema e como elas ajudam a realizarem suas tarefas, de forma simples e objetiva.

Assim, outros objetivo importante da PV é criar ferramentas eficazes com usabilidade, que se baseiem em expressões visuais e toda a tecnologia gráfica. E desta forma, ajudar usuários e programadores a realizarem seus trabalhos de forma rápida e produtiva, e conseqüentemente, com menos erros.

Entretando, um grande mal-entendido que esses fatos originam é que a PV tem como principal objetivo eliminar por completo o "texto" das linguagens. Isto é um grande engano, já que diversas linguagens de programação visual os incorporam, porém em um contexto multidimensional.

Estamos convencidos de que a PV e sua união com linguagens textuais caminham para uma forma mais clara de desenvolvimento de programas, mantendo a descrição e a abstração em alto nível e tornando-a mais legível por outras pessoas.

Um exemplo em nosso cotidiano seriam instruções sobre o caminho e um mapa. As instruções seriam como as linguagens textuais, muitas vezes confusas e complicadas de entender e lembrar. O mapa faria o papel da linguagem visual, com informações mais claras e precisas. Porém se unirmos as duas formas de descrição, teríamos um valor agregado muito maior e conseqüentemente uma maior compreensão por parte das pessoas.

Outra grande vantagem de uma LPV é a capacidade de retorno imediato (*feedback*) ao programador ou usuário. *Tanimoto* (5) descreve essa capacidade de oferecer retorno a qualquer edição do programa como *liveness (*responsividade do sistema) e a classifica em quatro níveis quanto ao *feedback*:

1. Nenhum retorno
2. Algum retorno, mas não imediato e automático
3. Retorno imediato e automático
4. Retorno imediato e automático, incluindo eventos de mouse, *ticks* de relógio, e outros dados que afetam o estado atual do programa.

Podemos também classificar algumas LPV quanto ao seu uso e suas características. Alguns exemplos são:

- Linguagens imperativas e por demonstração.
- Baseadas em exemplos.
- Baseadas em formulários e planilhas (fórmulas).
- Baseadas em regras (pré-condição e pós-condição).
- Controle de fluxo.
- Controle de dados.

Em resumo, temos que as principais vantagens da Programação Visual são:

- Figuras e imagens traduzem um significado mais eficiente do que palavras. Pensamentos são feitos em imagens, de forma multidimensional.
- São melhores para entender e serem lembradas, com alto nível de detalhes. Pessoas preferem imagens a palavras, exemplos concretos a abstratos. Também não impõem barreiras.
- A experiência em programar se torna mais agravável.
- Menos ênfase na sintaxe.
- Tempo de desenvolvimento e manutenção reduzidos, pois o código é mais fácil de ser gerado
- Custos de desenvolvimento e manutenção reduzidos.
- Concretude
- Manipulação direta de objetos
- Explicitação de relacionamentos.
- Retorno visual imediato. Mostra automaticamente os efeitos de uma edição em um programa (como as fórmulas em uma planilha de cálculo, que recalculam seus valores imediatamente quando a entrada também é modificada)
- Flexibilidade.
- Consistência.

7.1. Críticas, Problemas e Desafios

Apesar das inúmeras vantagens da Programação Visual e de Linguagens Visuais, algumas críticas pertinentes ainda são levantadas, colocando em discussão a sua real utilidade no universo da computação. Apresentaremos a seguir, em tópicos, alguns pontos de discussão de maior peso encontrados na nossa pesquisa:

7.1.1. Domínio Limitado

As LPV não foram convincentes e disseminadas até então, pois têm sido úteis apenas para domínios restritos e específicos de aplicação, principalmente para trabalho e pesquisas do meio acadêmico, deixando desconfiança sobre sua real utilidade em trabalhos reais e de maior escala.

Deste modo, temos que uma das barreiras a serem quebradas ainda é demonstrar que a Programação Visual pode ser usada para problemas do mundo real, e com isso descobrir novos e apropriados domínios nos quaispossamos aplicá-la.

Apesar disso, a Programação Visual tem demonstrado bastante sucesso e trazido grandes benefícios para as áreas que são aplicadas atualmente: programas de controle financeiro, construção de animações, simulações biológicas, depuração de programas. Acreditamos que a cada dia novas pesquisas surgirão com o objetivo de aumentar a gama de domínios aplicáveis à Programação Visual e provar que esta pode ser usada para ajudar usuários e programadores.

7.1.2. Resistência da comunidade

Ainda há muita resistência dos programadores em aderirem à computação visual. Eles partem do princípio de que a programação é fundamentada em conhecimento lógico e matemático e que não é possível uma pessoa sem o conhecimento da ciência da computação programar com sucesso. Fazendo uma analogia com um engenheiro mecânico: mesmo que você disponha de todas as peças de um carro separadamente, é quase impossível montá-lo sem o conhecimento de um engenheiro.

7.1.3. Escalabilidade

O problema de escalabilidade certamente deve ser o mais criticado pela comunidade resistente. O problema se concentra em como fazer uma aplicação expansível sem sacrificar os objetivos de melhor lógica e entendimento do programa. Ou seja, refere-se à capacidade das LPV´s em serem usadas por programas grandes. Atualmente a PV ainda apresenta restrições na construção de programas grandes ou que processam muitos dados.

Ignorando o problema de domínios de aplicação previamente discutido, podemos citar os três problema de escalabilidade: a limitação ao tamanho da tela do usuário, representação estática do programa e documentação.

Os elementos gráficos das LPV´s ocupam muito espaço na tela, ficando assim inviável a construção de programas grandes. Outro fator conseqüente é que não

podemos contar com a capacidade e técnica do usuário em organizar as informações e de aproveitar de forma otimizada os espaços disponíveis na tela. Peter Deutsch diz: "É impossível ter mais de cinqüenta elementos gráficos na tela ao mesmo tempo. Então como será possível construir um sistema operacional utilizando Programação Visual?".

Entretanto as linguagens textuais apresentam basicamente o mesmo tipo de problema enfrentado há muitos anos, pois é impossível visualizar, mantendo os níveis de leitura, mais do que quarenta a cinqüenta linhas de um programa escrito em texto na tela.

Isto pode impactar em outros subproblemas da escalabidade e afetar diretamente as vantagens da própria Programação Visual. Se a técnicas de visualização e organização não são bem desenvolvidas, o programa se torna pouco eficiente e sem a responsividade necessária, afetando-o em níveis mais baixos de *liveness*. Pesquisas focadas na melhora da navegação de interfaces, como as feitas na área de Interação Homem-Computador, e visualização de dados, podem ajudar muito as LPV´s a serem mais adaptáveis a programas de larga escala.

Como as LPVs dão menos ênfase à sintaxe, elas precisam de uma representação estática eficiente. A falta disso se torna uma barreira para as LPVs em termos de revisão, análise, entendimento e explicação dos programas construídos. Por exemplo, para um programa que deve conter dois retângulos alinhados, devemos colocar uma linha e em seguida construir os dois retângulos de forma alinhada, para só depois apagar a linha. Assim, programas que envolvem duas ou mais dimensões se tornam difíceis de serem construídos e de serem representados estaticamente, ou ficam com uma difícil compreensão. Uma medida a ser tomada é desenvolver LPVs com capacidade de esconder detalhes adicionais e excessivos em diferentes níveis de abstração.

Sabemos também que a documentação permite que o programador ofereça informações adicionais que não podem ser convertidas ou explicitadas no código. Entretando a documentação em linguagens visuais ocupam espaço na tela, influenciando ainda mais o problema de falta de espaço e limitação da tela. Ainda assim temos o problema da ineficiência da maioria dos programadores em escrever uma boa documentação. Muitos optam em documentar pouco, o com pouca exatidão. Programas com maior responsividade e/ou com uma boa ferramenta de depuração acabam ajudando os usuários e outros programadores a entenderem o fluxo, as propriedades e como o programa é executado, aumentando a consistência entre o código e a informação da documentação, mesmo que seja pouca.

7.1.4. Poucas ferramentas e falta de eficiência

Atualmente existem poucas ferramentas para PV que implementam de forma eficiente as linguagens visuais. A maioria ainda possui alguma deficiência em atributos já discutidos aqui (como depuração, teste, responsividade, etc.) considerados essenciais em computação. Alguns programas chegam a demoram muito tempo para terminar a execução, principalmente em sistemas de maior responsividade que

demandam uma maior eficiência em redesenhar telas para mostrar constantes modificações no estado do programa.

7.1.5. Abstração Procedural

A habilidade de criar métodos e rotinas que encapsulam os detalhes de subtarefas foi um grande avanço para a programação. Este é um fator tão importante para a computação, que hoje em dia é difícil pensarmos em programar sem utilizar tal abstração. Entretanto, a maioria das LPVs ainda não implementam a abstração procedural. Algumas já a fazem de modo que permitem que o programador selecione parte do código e criem um componente novo na biblioteca de funções, promovendo a reusabilidade de componentes.

7.1.6. Abstração e Encapsulamento de Dados

Assim como a abstração procedura, a abstração de dados é igualmente importante para encapsular informações e manter a consistência, visibilidade e interatividade, permitindo o acesso a alguns dados apenas por meio de operações pré-definidas pelo programador

7.1.7. Checagem de tipos de dados

A checagem dos tipos de dados é feita para detectar erros de incompatibilidade em determinadas operações. Apesar da maioria das LPVs permitirem que isso seja feito, algumas ainda apresentam problemas em atingir objetivos de simplicidade e responsividade. Algumas das técnicas abordadas pela maioria das LPVs são: a checagem estática de tipos explícitos e implícitos, como também a checagem dinâmica dos dados (para maior flexibilidade e simplicidade).

7.1.8. Persistência

A persistência de dados é essencial para programas de grande porte. Sem ela, os dados não podem ser mais acessados após o término da execução do programa, o que é inviável para a maioria das aplicações. A persistência em algumas LPVs é muito fraca ou até nula. Apesar disto, outras apresentam ferramentas de persistência, basicamente das seguintes formas:

- Explicitação de arquivos para entrada e saída.
- Incorporação de linguagens para bancos de dados.
- Todos as informações e dados são persistidos (como em planilhas).

7.1.9. Ferramentas além-programação

A maioria das LPVs não possui ferramentas de uso fundamental para o desenvolvimento de sistemas, sejam de pequena ou grande escala. Ferramentas de controle de versão, compilação condicional, geração de bibliotecas e portabilidade, por exemplo, não aparecem em quase nenhuma VPL

7.1.10. Transparência da interface

A maioria das interfaces de ambientes para desenvolvimento de programas utilizando PV possuem deficiência na transparência, fazendo com que a entrada de dados e a construção de diagramas complexos necessitem de um planejamento prévio

que requer algum tempo e conhecimento por parte do programador da interface, deixando de lado a simplicidade prometida pelas LPV.

Outro ponto a ser discutido seria a naturalidade em representar visualmente certas estruturas de fluxo e/ou controle no programa. Por exemplo, para se fazer um laço, uma linguagem textual é muito mais apropriada por fazer apenas o uso de uma palavra, como *while* ou *for*, do que a construção de um ícone específico, ou um grafo cíclico para tal representação.

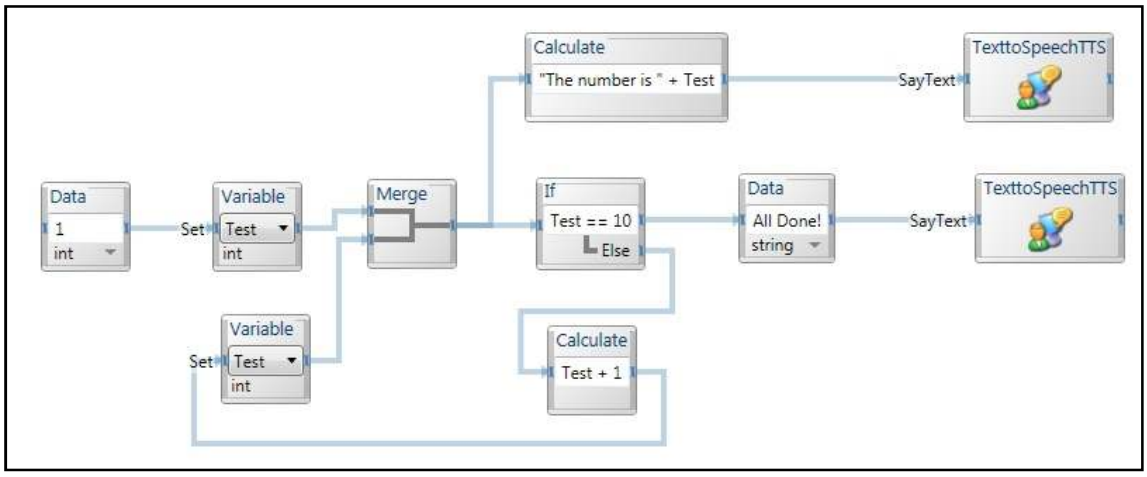

Figura 15: Construção de um ciclo utilizando a LPV da Microsoft Robotics.

7.1.11. Práticas de Programação

Algumas LPVs são criticadas por não usarem boas práticas de programação. A maioria faz o uso de *go-tos*, o uso apenas de variáveis globais, abstração procedural, documentação e comentário (algumas já discutidas anteriormente).

8. Estudo de Caso

Como dito anteriormente, no estudo de caso vamos primeiramente nos aprofundar um pouco mais no estudo do pacote Maiges Pack:

8.1. Maiges Pack

8.1.1. Introdução

O pacote foi desenvolvido no projeto MAIGES (*Mathematical Analysis of Interacting Gene Expression Systems*) com o objetivo de oferecer ferramentas de análise de microarray, garantindo reprodutibilidade e facilidade na documentação do processo adotado. Essa necessidade existe pois diversas abordagens de análise podem ser utilizadas de acordo com o tipo de estudo que está sendo feito.

8.1.2. Estrutura do pacote

O MaigesPack integra vários pacotes do R afim de realizar diversos tipos de análise de dados de microarray e ainda agrega algumas funcionalidades que facilitam o processo como um todo. Entre essas facilidades estão a verificação em tempo de execução da qualidade dos dados experimentais, a capacidade de geração de gráficos e exportação de resultados em formato HTML. Alguns dos pacotes que o MaigesPack integra vêm do site do CRAN (amap, class, e1071, gplots, MASS, R2HTML, rgl e som) e outros vêm do projeto Bioconductor (annotate, convert, graph, limma, marray, multtest e OLIN).

O problema principal na integração de todos esses pacotes é que a maioria dos pacotes desenvolvidos utilizam estruturas de dados bastante específicas, tornando difícil a comunicação entre procedimentos diferentes implementados em pacotes diferentes.

Após analisar a estrutura de classes do pacote foi montado um diagrama (Figura 16) em que é possível verificar que o fluxo de execução dos métodos e funções empregados em um processo de análise segue a estrutura de classes do pacote. Nesse diagrama cada classe está representada dentro de um retângulo que possui o nome, funções, métodos e atributos dessa classe; as arestas representam métodos computacionais que transformam um objeto da classe que a aresta sai em um objeto da classe em que a aresta chega. Essa característica do pacote confere modularidade ao processo de análise, já que cada processo de análise pode ser definido como uma seqüência de métodos específicos. Ainda conseguimos uma maior facilidade na extensão do pacote, já que é possível produzir novos módulos de análise que seriam facilmente incorporados ao pacote.

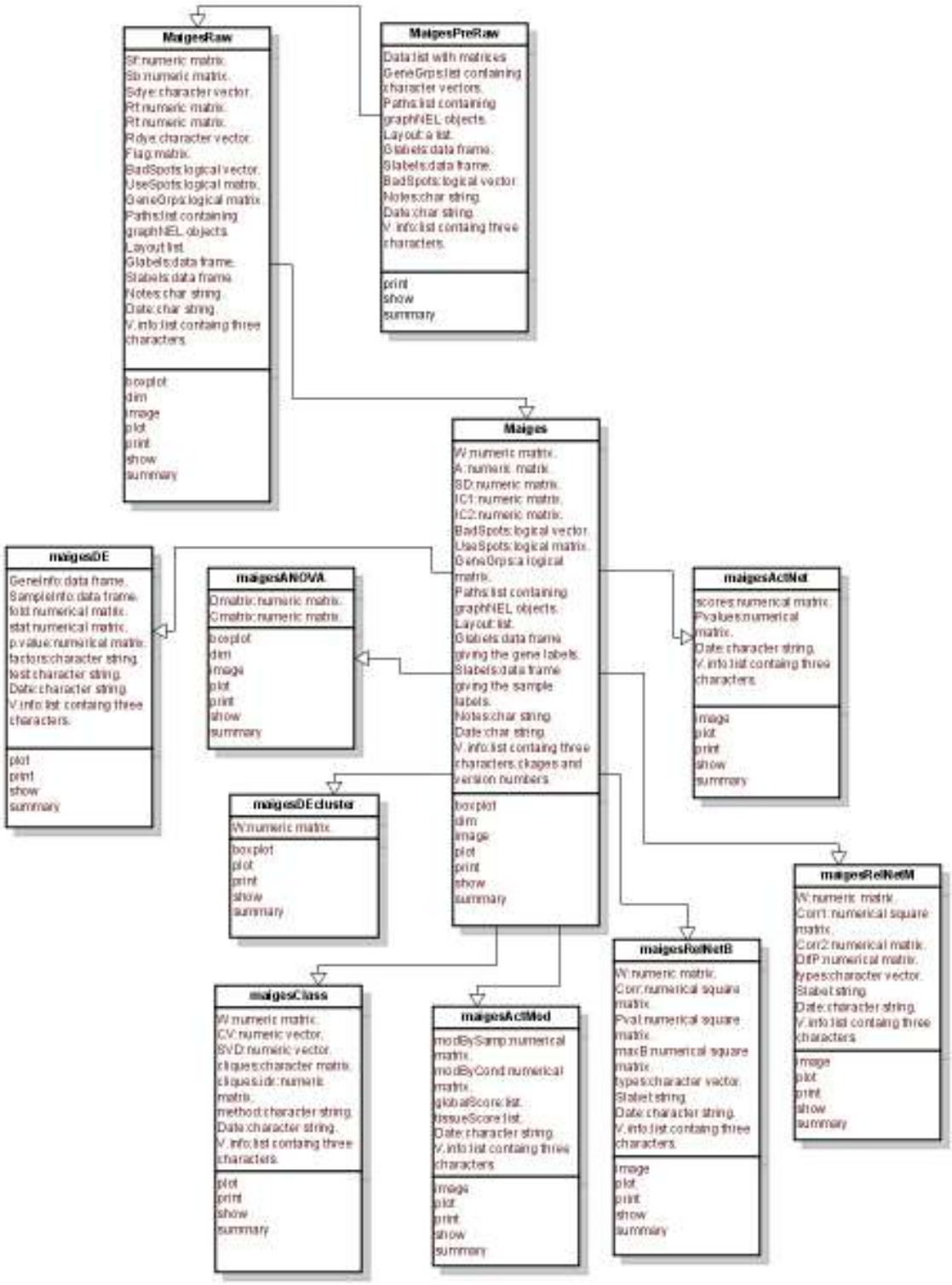

Figura 16: Diagrama de classes do pacote MAIGES.

8.2. Funcionamento do pacote

Para qualquer processo de análise de dados de microarray que seja feito utilizando esse pacote é necessário seguir 4 passos:

1 - Ler os dados
2 - Realizar análise exploratória dos dados para verificar a qualidade
3 - Normalização
4 - Aplicar os métodos necessários dependendo do que se quer responder

O início de um processo de análise de microarray se inicia com a carga dos dados (passo 1), que é executada pelo método *load.data.* Este, gera um objeto da classe *maigesPreRaw* - a classe de entrada dos dados no R. A partir dessas informações iniciais, é possível diagnosticar eventuais erros de leitura ou viés.

Depois do objeto criado, pode ser transformado em um objeto da classe *maigesRaw* com o método *create.maigesRaw*. Nessa fase do processo, os objetos armazenam os dados brutos de um experimento de microarray.

Nesse caso, o objeto pode ser convertido em um objeto das classes *marrayRaw* e *marrayNorm* ou *RGList* e *MAList* - as duas primeiras classes estão definidas no pacote *Marray* e as outras duas no pacote *Limma*, esses pacotes possuem algumas funções específicas para análise exploratória de dados (passo 2) que podem ser interessantes dependendo do experimento que esteja sendo feito. Nessa fase também é possível gerar um objeto da classe *maiges* empregando uma das funções de normalização disponíveis no pacote.

Após usar uma das funções de normalização (passo 3), geramos um objeto da classe central do pacote (*maiges)*. Todos os métodos de análise de dados estão implementados para atuar sobre objetos dessa classe.

Nesse momento do processo é possível tomar diversos rumos dependendo do estudo que esteja sendo feito (passo 4), o diagrama do pacote *maiges* mencionado anteriormente deixa bem claro a diversidade de caminhos a serem seguidos nessa etapa.

As classes abaixo são destinadas a armazenar resultados e executar métodos de análises finais, finalizando o processo de análise de dados:

- A classe *maigesANOVA* é uma extensão da classe *maiges* que contém outras duas matrizes utilizadas para o ajuste de um modelo de análise de variância.

- A classe *maigesDE* armazena resultados de análises que buscam genes diferencialmente expressos.

- A classe maigesDEcluster é uma extensão da classe maigesDE em que métodos de agrupamento podem ser executados.

- A classe maigesClass armazena resultados de análises de descriminação ou classificação.

- A classe maigesActMod armazena resultados de análises de classificação funcional de grupos gênicos.

- A classe maigesRelNetB armazena resultados de análises de redes de relevância.

- A classe maigesRelNetM armazena resultados de análises de redes de relevância usando um modelo aprimorado.

- A classe maigesActNet armazena resultados de análises de classificação funcional de redes de regulação gênica.

Após a execução desses 4 passos temos uma noção completa de como ocorre uma análise de dados de microarray do início ao fim utilizando esse pacote. Todo esse processo é mais complicado do que pode parecer, vamos apresentar agora alguns dos problemas enfrentados pelos usuários do R.

Para gerar e manipular os objetos do R é necessário algum conhecimento de programação e para cada função que vai ser aplicada é necessário que se conheça a sua assinatura e seus parâmetros. Além disso, apesar de toda a preocupação que o CRAN e o Bioconductor têm com a documentação, essa nem sempre é impecável e ás vezes pode ser bastante difícil para o usuário entender como fazer determinados pacotes rodarem corretamente.

8.3. Execução do Tutorial

Para que o leitor tenha uma noção melhor desses problemas vamos mostrar passo a passo como seria para um usuário trabalhar com o pacote Maiges. Vamos reproduzir abaixo a execução de uma parte do tutorial do pacote maiges que pode ser encontrado em:

http://www.bioconductor.org/packages/2.1/bioc/vignettes/maigesPack/inst/doc/maigesPack_tutorial.pdf

Com o R já instalado, após baixar o pacote Maiges e instalá-lo no R chegamos no primeiro comando do tutorial que é:

```
library(maigesPack)
```

Esse comando é responsável pelo carregamento do pacote depois de instalado. Ao digitar esse comando e apertar “Enter” o R informa o primeiro empecilho, os pacotes que o Maiges usa não estão instalados e não é possível continuar a execução do tutorial. Para sanar esse problema tive que baixar todos esses pacotes e instalar

corretamente. Feito isso recomecei, o comando “library(maigesPack)” agora rodou, todos os outros pacotes foram carregados e eu posso continuar a execução. Partimos para o segundo comando:

```
data(gastro)
```

Esse comando faz o carregamento do objeto gastro. Não tivemos nenhum problema com ele e passamos para o próximo comando que é responsável por carregar os valores numéricos no R:

```
gastro = loadData(fileConf = "load_gastro.conf")
```

Ao enviar o comando, o R responde que está tendo problemas com a função de leitura de arquivos e que não consegue encontrar o arquivo especificado. Depois de algum trabalho para entender o erro, percebi que o problema estava no caminho indicado.

Substituí o comando pelo abaixo para funcionar:

```
gastro = load.data(file.conf = "C:/Arquivos de programas/R/R-
2.5.1/library/maigesPack/doc/gastro/load_gastro.conf")
```

Os dois comandos seguintes não apresentaram problemas. Depois deles chegamos no comando que transforma o objeto da classe maigesPreRaw em um objeto da classe maigesRaw:

```
gastro.raw = createMaigesRaw(gastro, greenDataField = "Ch1.Mean",
+ greenBackDataField = "Ch1.B.Mean", redDataField = "Ch2.Mean",
+ redBackDataField = "Ch2.B.Mean", flagDataField = "Flags",
+ gLabelGrp = "GeneName", gLabelPath = "GeneName")
```

Esse comando rodou corretamente também, porém é bem complicado entendê-lo e, se alguma alteração nos parâmetros tiver que ser feita, é possível que o trabalho seja bem grande. Depois dele chegamos no comando que plota o gráfico na tela, como indicado na Figura 17.

```
plot(gastro.raw[, 1], bkgSub = "none")
```

Nessa altura da execução do tutorial creio que já ficou bem claro para o leitor o tipo de dificuldade que é encontrado pelos usuários desse pacote e então podemos parar a sua execução. Foi justamente pensando nesse tipo de dificuldade que procuramos facilitar a vida do usuário. Sabemos que muito esforço já foi feito pelos desenvolvedores do R e de seus pacotes e sabemos que a interação entre o R e os seus usuários ainda não é tão simples quanto se deseja.

Certamente seria um grande desperdício de tempo abrir mão de tudo que já foi feito e recomeçar do zero. Nesse momento nos perguntamos como poderíamos então aproveitar tudo o que já foi desenvolvido e mesmo assim facilitar para o usuário final.

É nesse contexto que se insere o Kepler, um software de programação visual que pode ser integrado ao R de que vamos falar um pouco mais.

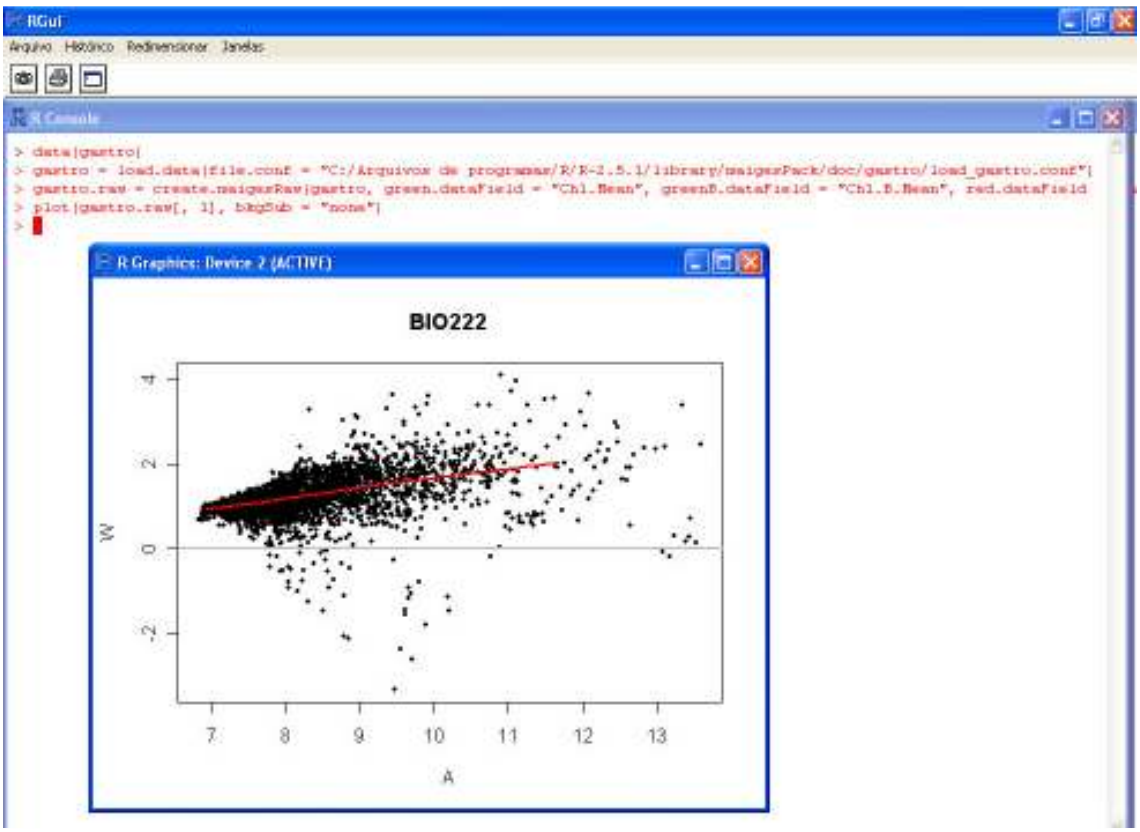

Figura 17: Execução do tutorial do pacote maiges no R.

9. Kepler

9.1. O que é o Kepler?

O Kepler é um programa de computação visual baseado em *workflows* amplamente utilizado por cientistas. Estes, conseguem criar e executar *workflows* científicos que podem englobar processos complexos, integrando programação R, MATLAB, acesso a banco de dados externo, entre outras funcionalidades. O software é baseado em componentes chamados de “atores”, que comunicam-se entre si através de “portas” de entrada e saída. Desta forma, eles conseguem trabalhar de forma encapsulada. Cada "Ator" faz ações tais como execução de código, acesso a dados, processamento de dados.

9.2. Funcionamento do Kepler

O sistema foi desenvolvido a partir do sistema Ptlemy II, que controla a execução de um workflow por meio dos diretores. Neste sistema, um ator pode ter multiplas portas de entradas e saída de dados. Ptolemy também é capaz de executar o workflow de forma estática ou dinâmica, com alterações no tempo de execução. Baseado neste sitema, O Kepler acabou herdando estas características.

A base principal do Kepler é a utilização dos atores, que são responsáveis por:

- Encapsular as ações.
- Canalizar as entradas, executar e fornecer saída.
- As portas são a forma centralizada de comunicação entre os atores. Os dados são heterogêneos e podem ser recebidos e enviados para:
    - banco de dados
    - saídas em arquivos, saídas gráficas e bloxplots
    - páginas web
    - FTP e SSH

Os diretores são responsáveis por gerenciar a execução dos atores. Isso permite que o workflow seja executado de forma contínua, única, ou seguindo regras personalizadas.

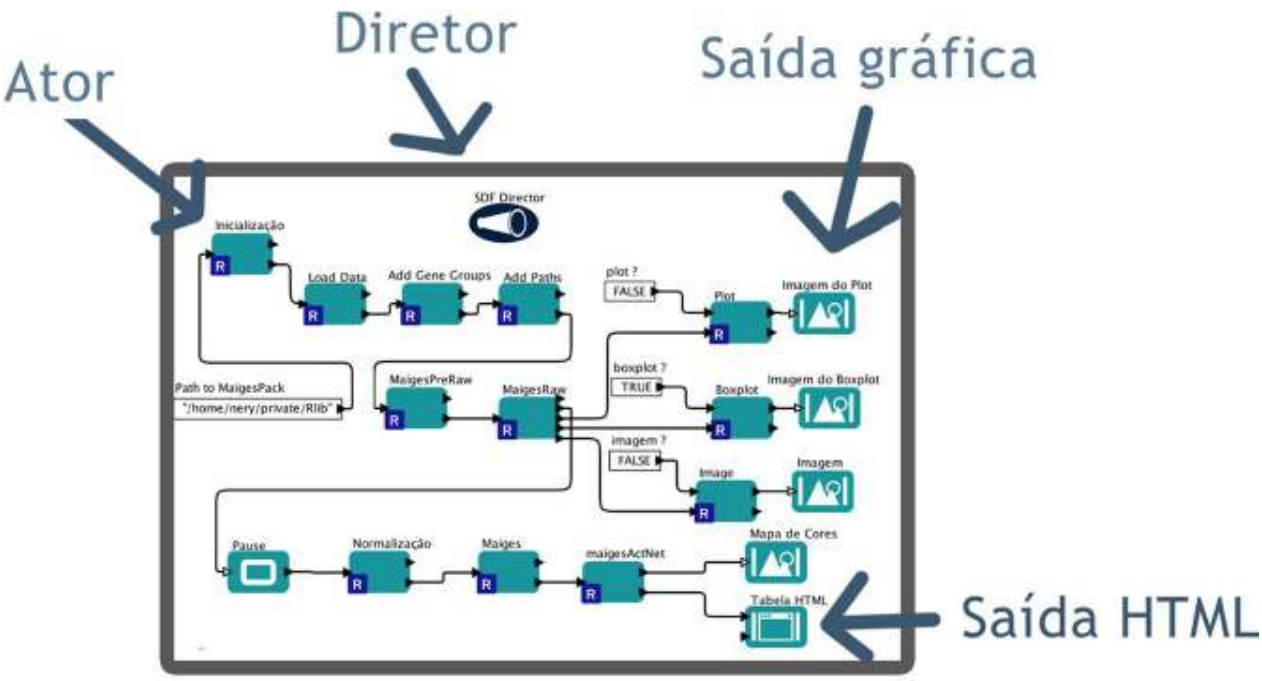

Figura 18: Implementação no Kepler do pacote Maiges – Classificação funcional de redes gênicas.

Nosso intuito inicial era (como na imagem acima) implementar no Kepler todos os caminhos de execução possíveis de uma análise de microarray feita utilizando o pacote maiges. Dessa maneira um usuário que não possui experiência com o R e que desejasse utilizar as funcionalidades existentes no pacote maiges poderia apenas entrar com os dados que desejasse, configurar (de forma visual) o caminho que sua análise deveria seguir e "apertar play". O Kepler se encarregaria de toda a criação dos objetos, chamada de funções e etc.

Como pudemos verificar, o Kepler possui algumas limitações que impedem o seu funcionamento ideal para integração com o R. De modo a impossibilitar essa integração. Trataremos melhor esse assunto em um capítulo a seguir.

10. Workflows Científicos

Análise de dados científicos utiliza grande volume de informações e são realizadas em etapas heterogêneas, que podem envolver processos de normalização, acesso a banco de dados, acesso a repositório de informações, análise e saída de dados. Frequentemente esse tipo de análise não segue uma regra genérica de execução, fazendo com que os processos sejam sempre alterados. A utilização de uma ferramenta capaz de automatizar e unificar essas análises, economiza tempo e facilita sua execução.

Workflows científicos são ferramentas flexíveis para executar análises de dados geralmente complexas, incluindo dados científicos. Cada workflow é organizado em diversas etapas chamadas de "atores", que quando executados na ordem correta, produzem saídas gráficas ou textuais. As etapas são representadas por digrafos (grafos dirigidos) e cada nó deste grafo representa a execução de uma tarefa.

Entre suas principais vantagens estão sua fácil reusabilidade, integração com diversas tecnologias, redução dos custos de de execução. Em geral, análise de dados científicos utiliza grande volume de dados que são obtidos de diferentes fontes. Um workflow científico é capaz de utilizar dados dessas diversas fontes, agilizando o processo de execução.

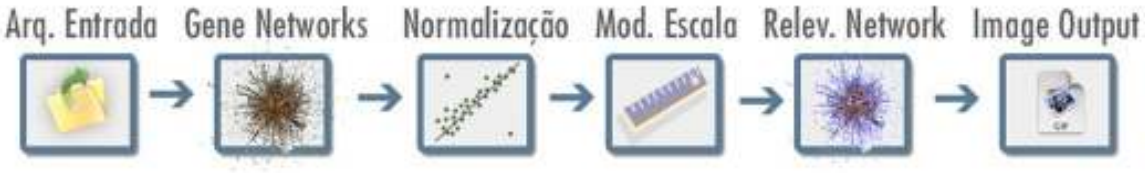

Figura 19: Exemplo de workflow científico que recebe dados científicos, analisa e gera uma saída gráfica.

11. Avaliação sobre a adequabilidade

Depois de conhecer o funcionamento das Linguagens de Programação Visual, Kepler e R, podemos finalmente discutir a adequabilidade do Kepler para o pacote Maiges.

A idéia de utilizar PV parece perfeita na análise de *microarray*, especialmente por ser um processo heterogêneo. Como vimos acima, o diagrama final em um processo de análise de *microarray* pode trilhar diversos caminhos.

O grande gargalo que encontramos especificamente na utilização do Kepler, é o fato de que a comunicação entre os atores é feita de forma limitada. As portas (entrada e saída de dados) aceitam apenas dados simples: booleanos, numéricos, matrizes, entre outros. Isso passa a se tornar uma limitação quando pensamos que os códigos que queremos utilizar dentro do Kepler não foram desenvolvidos com este propósito.

O pacote Maiges não é exceção. Ele foi projetado para ser rodado dentro do R, e não no Kepler. As entradas e saídas de cada passo de execução deste pacote são objetos e classes inteiras, inclusive seus dados. Desta forma, fica inviável a utilização do Kepler da forma usual.

Uma forma de contornar este problema seria manter uma instância aberta do R, dentro do Kepler. Isto se faz necessário porque o Kepler inicia e finaliza uma instância do R em cada ator, perdendo os dados. Conversando com o desenvolvedor responsável pelo funcionamento do R dentro do Kepler, Dan Higgins, compreendemos que não é possível manter esta instância aberta.

Consequentemente, nossa solução se torna muito ineficiente, pois utilizamos o R em quase todos os atores. Iniciar e finalizar uma instância de R é muitos custoso.

Uma solução possível para nosso problema é salvar o workspace do R em cada ator do Kepler e utilizá-lo no próximo ator. Isso se torna inviável, pois trabalhamos com volume enorme de dados. Não conseguimos medir o tempo que demora para salvar e abrir um workspace nessas circunstâncias, apesar de termos tentado simulação com dados reais.

Tendo em vista que a análise e processamento de dados de bioinformática requer muito recurso de hardware, esta solução seria muito ineficiente. Uma idéia por trás da programação visual é unir programas e dados heterogêneos, que não foram criados com a intenção de serem rodados por plataformas de programação visual. Seria possível programar em R de tal forma que fosse possível utilizar o Kepler, mas esta não é a solução para o nosso problema pois os pacotes de análise em R já foram criados.

Para as próximas versão do Kepler, podem haver modificações no RExpression para que seja possível a utilização de pacotes mais complexos em R. Criar uma maneira de manter a instância aberta de R também pode ser interessante, apesar de perder a idéia de encapsulamento dos atores.

12.Conclusão

A demanda por sistemas computacionais na área de científica aumentou nos últimos anos. Porém, o desenvolvimento de programas específicos para tais tarefas não acompanha este crescimento, pois é necessário conhecimento em programação. Por existir este obstáculo para o mundo científico, resolvemos focar nosso trabalho na pesquisa de soluções para tornar a programação mais fácil para o usuário comum, de modo a quebrar a barreira.

Primeiramente, começamos tendo o contato e estudando os conceitos de bioinformática, Linguagens de Programação Visual (conceito, limitações e vantagens), tese de doutorado de Gustavo Henrique Esteves e o projeto Maiges, programação R. Com esse estudo, percebemos a dificuldade encontrada por pessoas da área da biologia na análise de *microarray*.

Por outro lado, para nós programadores, realizar alguma refatoração do código (idéia principal do projeto) ou desenvolver novas ferramentas para o sistema, se tornava algo desgastante pois nos faltava conhecimento do processo de análise de *microarray*. Após analisar algumas vezes o sistema buscando melhorias , sentimos a necessidade de tornar o sistema mais amigável para o usuário final. Afinal, os biólogos que necessitassem fazer suas análises, gostariam de algo simples para alcançar seu objetivo sem a necessidade de ter que entender o lado computacional.

Neste cenário surge a idéia de implementar o Maiges no sistema de programação visual Kepler. Precisaríamos então apenas utilizar o código em R do pacote Maiges para a construção do workflow no Kepler, de modo que ficasse algo intuitivo e simples para quem quisesse utilizar, livrando assim os usuário da linha de comando. Como vimos neste estudo, são inúmeras as vantagens da utilização de workflows científicos na área de biologia.

Começamos estudando o funcionamento do programa, desenvolvendo programas simples e entendendo alguns exemplos encontrados para um maior entendimento das ferramentas do sistema. A partir daí, o foco do trabalho havia se desviado da idéia original, porém com tema ainda voltado à bioinformática. Passaríamos assim a analisar a adequabilidade do Kepler para a implementação de um pacote de bioinformática feito em R, especificamente o Maiges.

Conforme descrevemos acima, o Kepler não é o programa ideal para implementação do pacote Maiges. Isto mostrou para nós que, apesar do Kepler ser muito útil para o meio científico constatado por nós pela construção e execução de diversos workflows nas mais variadas áreas de conhecimento, este se tornou inviável e ineficiente para a construção do pacote Maiges com o objetivo de tornar este mais transparente e simples para os usuários.

Isso não exclui a possiblidade de utilizarmos sistemas de programação visual na análise de *microarray*. Apenas ilustra um problema real encontrado na integração entre biologia e computação. Podem existir pacotes de bioinformática que podem ser

utilizados no Kepler, como também podem existir outros sistemas de programação visual que conseguem implementar o Maiges.

13. Bibliografia

[1] Brad A. Myers. "Taxonomies of Visual Programming and Program Visualization", September 1989.

[2] Najork, Marc A, Eric J. Golin, "Enhancing Show-and-Tell with a polymorphic type system and higher-order functions..", October 1990.

[3] Citrin, Wayne. "HCI Issues in Visual Programming", Journal of Visual Languages and Computing, 1996.

[4] Smith, D. C. "*Pygmalion: A Computer Program to Model and Stimulate Creative Thought*". Stuttgart, Basel. 1977.

[5] S. Tanimoto, *"VIVA: A visual language for image processing"*. 1990

[6] J.Webster (Ed.), *"Wiley Encyclopedia of Electrical and Electronics Engineering. ",* 1999 John Wiley & Sons, Inc

[7] Wei Yu, "Visual Programming – Component for programmers".

[8] - Hils 92
Daniel D. Hils, "A Visual Programming Language for Visualization of Scientific Data", Deparment of Computer Science, University of Illinois at Urbana-Champaign, 1992

[9] Gustavo Henrique Esteves, "Métodos estatísticos para a análise de dados de cDNA microarray em um ambiente computacional integrado", Universidade de São Paulo, 2007

[10] "Kepler: An Extensible System for Design and Execution of Scientific Workflows", Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludäscher, Steve Mock

[11] Glória Regina Franco, "Uso de Microarray para análise da expressão gênica diferencial", Departamento de Bioquímica e Imunologia Universidade Federal de Minas Gerais

[12] Genepix Pro 6.0
http://www.moleculardevices.com/pages/software/gn_genepix_pro.html

[13] Kepler Project
http://kepler-project.org/

[14] R Project
http://www.r-project.org/

[15] "Provenance Collection Support in the Kepler Scientific Workflow System", Ilkay Altintas, Oscar Barney, Efrat Jaeger-Frank

[16] Wikipedia: Bioinformatics workflow management systems
http://en.wikipedia.org/wiki/Bioinformatics_workflow_management_systems

[17] ”Tecnologia de Microarrays de DNA”, Michael Waisberg, MD, MSc
biodados.icb.ufmg.br/sebio/Introducao_do_curso.ppt

[18] “Uso de Microarray para analise de expressao genica diferencial”, Glória Regina Francoc
http://biotec.icb.ufmg.br/cabi/aulas/aula13.pdf

[19] Pat Bron’s Lab, Department of Biochemistry – Stanford University
http://cmgm.stanford.edu/pbrown/

[20] "The Living Genome", Patrick O. Brownc
http://www.hhmi.org/research/investigators/brown.html

[21] “The MGuide. Version 2.0 – Build your own arrayer”
http://cmgm.stanford.edu/pbrown/mguide/index.html

[22] “Tecnicas de Aprendizado de Maquina para Problemas de
Biologia Molecular”, M. C. P. de Souto, A. C. Lorena, A. C. B. Delbem, A. C. P. L. F. de Carvalho

[23] Wikipedia: Affymetrix
http://en.wikipedia.org/wiki/Affymetrix

[24] “Técnicas de Análise de Expressão Gênica em Eucariotos”, Celso E. Benedetti

[25] “GeneChip® Microarrays - Activity #3 – Manufacturing of GeneChip Microarrays and Building Models”
http://www.affymetrix.com/corporate/outreach/lesson_plan/downloads/student_manual_activities/activity3/activity3_manufacturing_background.pdf

[26] “Comparison of methods for image analysis on cDNA microarray data (2000)”, Yee Hwa Yang, Michael J. Buckley, Sandrine Dudoit, Terence P. Speed

[27] “Image Analysis and Recognition: Second International Conference, ICIAR 2005”, Luis Rueda and Li Qin

[28] “A New Method for DNA Microarray Image Segmentation”, Luis Rueda and Li Qin

[29] “Evaluating the performance of microarray segmentation algorithms”, Antti Lehmussola, Pekka Ruusuvuori and Olli Yli-Harja

[30] “Efficient Gridding of Real Microarray Images”, Giuseppe Lipori

[31] “Microarray image addressing based on the Radon transform”, Antoniol, G.; Ceccarelli, M.; Petrosino, A.;

[32] “A Markov Random Field Model of Microarray Gridding”, Mathias Katzer; Franz Kummert; Gerhard Sagerer

[33] “Adaptive techniques for microarray image analysis with related quality assessment”, Sebastiano Battiato; Gianpiero Di Blasi; Giovanni Maria Farinella; Giovanni Gallo; Giuseppe Claudio Guarnera

[34] “Microarray Image Analysis, Lucia Maddalena”, Lucia Maddalena

[35] “Genepix Pro 6 Microarray Image Analysis” http://www.moleculardevices.com/pages/software/gn_genepix_pro.html

[36] “Unsupervised technique for robust target separation and analysis of DNA microarray spots through adaptive pixel clustering.”, Bozinov, D. and Rahnenführer, J. (2002)

[37] “Ratio-based decisions and the quantitative analysis of cDNA microarray images.”, Chen, Y., et al. (1997)

[38] “An Introduction to Programming Robots with Microsoft Robotics Studio”, Sara Morgan Rea