

UCSF

UC San Francisco Electronic Theses and Dissertations

Title

Quantitative Pathway Modeling and Analysis in Cancer

Permalink

<https://escholarship.org/uc/item/4sr3z9tf>

Author

Novak, Barbara Anna

Publication Date

2007-06-15

Peer reviewed|Thesis/dissertation

Quantitative Pathway Modeling and Analysis in Cancer

by

Barbara Anna Novak

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Biological and Medical Informatics

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, SAN FRANCISCO

Copyright 2007
by
Barbara Anna Novak

Acknowledgements

This thesis marks the end of a long journey that began with the desire to understand the inner workings of life and resulted in the decision to pursue a career in bioinformatics. Along the way, many people have helped me find my path and sustained me and it is with great pleasure that I acknowledge their contributions.

I would like to thank my Ph.D. advisor, Ajay Jain, for all of his guidance and support. Throughout my time as a graduate student, he provided encouragement and helpful advice. More importantly, he taught me to look at problems scientifically, to rigorously examine my assertions and to present my research coherently. Without his support, I wouldn't be where I am today.

Without data to analyze and pathways to parse, this thesis wouldn't have been possible. I would like to thank all of the researchers who kindly shared their data with me: Frank McCormick, for the picture of RTK pathway that started it all; Joe Gray and the members of his laboratory for the breast and ovarian data sets; Fred Waldman, for the bladder data set; and Martin McMahon and Stefan Gysin, for the pancreatic data set.

I would also like to thank all of the people at caBIG, the Cancer BioInformatics Grid project at the National Cancer Institute, both for supporting this work and for helping me integrate my project with the greater community of pathway analysis research. I would especially like to thank Gary Bader, for his help with understanding the BioPAX standard, and Shannon McWeeney and Ted Laderas at Oregon Health Sciences University, for their work on rigorously testing QPACA's functionality.

I am thankful to all of the other members of the Jain laboratory for providing both support and needed distraction throughout the years. First, the other graduate students provided a stimulating and fun environment: Chris Kingsley (who shared my experience battling with caBIG), Lawrence Hon (who shared an office with me for many years and put up with my occasional desires to procrastinate), and Tuan Pham (who brought tasty Vietnamese sandwiches and kept me company when the other two had left for bigger and better pastures). Additionally, I would like to thank the two postdocs in the lab: Jane Fridlyand (for putting up with silly statistics questions) and Taku Tokuyasu (for making me think about the biology behind everything).

Without the help of the various program coordinators for the Biological and Medical Informatics program, particularly Barbara Paschke and Rebecca Brown, I would have been lost within the UCSF bureaucracy.

I am grateful to all of my friends for putting up with my attempts to explain what I do and for dragging me out to movies and dinners when I was immersed in work.

I would like to thank my parents, Blanka and Miloslav Novak, for standing by me and supporting me throughout my time as a student.

Lastly, and most importantly, I wish to thank my husband, Ellis Verosub, for patiently continuing to support and encourage me despite the lengthening tenure of my time as a graduate student. Without him, I would have been lost and to him I dedicate this thesis.

Abstract

Quantitative Pathway Modeling and Analysis in Cancer

Barbara A. Novak

Biological pathways describe the inter-relationships of genes, proteins, and molecules within a biological system. The study of specific pathways often requires painstaking research into each possible element of the network. Consequently, existing knowledge of gene regulatory networks is limited, resulting in hypotheses that are incomplete or at various levels of refinement. Computational modeling and quantitative analytical approaches seek to fill this information gap.

This dissertation describes the development of QPACA (**Q**uantitative **P**athway **A**nalysis in **C**ancer), a system for pathway visualization and analysis. QPACA addresses three aspects of the general problem: 1) representation and visualization of pathways in the context of biological data, 2) recognition of gene sets that are part of a pathway or coordinated process, and 3) augmentation of pathways by prediction of pathway membership. The pathway representation is designed to be flexible and extensible in order to enable the widest variety of pathway structures and components possible, while the analytical methods directly address the issues inherent in analysis of human systems without making limiting assumptions about the structure of pathways or discretizing data.

QPACA has been used to analyze a number of microarray data sets, employing both yeast and human samples. Four primary results are presented, each of which derives from aspects of QPACA's application to microarray data for analysis or visualization: 1) statistical analysis of differential expression patterns in the context of pathway

representations supports the generation of biological hypotheses; 2) gene expression data are sufficient to support computational recognition of hypothesized pathway gene sets across a broad variety of biological processes; 3) gene expression data can be used to produce ranked lists of gene products that are enriched for proteins that interact with members of a predefined pathway; and 4) the surprising ubiquity of detectable signals in expression data that bear on human pathway structure appears to be due to largely non-annotated transcriptional programs present within established pathways.

Table of Contents

Chapter 1 Introduction.....	1
1.1. Systems Biology	1
1.2. Modeling Pathways.....	3
1.3. Transcriptional Regulation Biology.....	4
1.3.1. Motif finding.....	6
1.4. Microarray technologies in cancer research.....	6
1.5. Statistical and other issues	9
1.5.1. Statistical issues of large data sets	10
1.5.2. Issues with heterogeneous data sets.....	12
Chapter 2 Review of Pathway Literature.....	14
2.1. Introduction.....	14
2.2. Pathway representation and curation	14
2.2.1. Pathway exchange formats	16
2.3. Pathway-based data analysis.....	16
2.4. Model recognition and induction	17
2.5. Conclusion	18
Chapter 3 QPACA: System Design	20
3.1. Introduction.....	20
3.2. Pathway model.....	21
3.3. QPACA optimization method.....	25
3.4. System architecture	29
3.4.1. Design overview	29
3.4.2. Implementation	30
3.4.3. QPACA toolset design.....	34
3.5. Conclusion	35
Chapter 4 Pathway Exploration and Visualization	36
4.1. Introduction.....	36
4.2. Exploration of gene/gene relationships in an array-based comparative genomic hybridization experiment	39
4.3. Exploration of CGH phenotype of ERBB2 in expression of RTK genes.....	41
4.4. Exploration of pancreatic cancer cell line data in the context of the RAS subpathway	44
4.5. Conclusion	47
Chapter 5 Computational Recognition of Pathways from Microarray Expression Data	48
5.1. Introduction.....	48
5.2. Materials and Methods.....	50
5.2.1. Data and data processing	50
5.2.2. Permutation testing and p-value calculation.....	51

5.3. Results.....	52
5.4. Discussion and conclusion.....	56
Chapter 6 Computational Prediction of New Pathway Members from Microarray Expression Data	57
6.1. Introduction.....	57
6.1.1. Cross-validation in biological pathway augmentation.....	58
6.2. Materials and Methods.....	59
6.2.1. Augmentation algorithm and scoring metrics.....	59
6.2.2. Data and data processing	60
6.3. Cross-validation of pathway augmentation using human and yeast microarray data	61
6.4. Cross-validation of pathway augmentation using closely related subpathways	65
6.4.1. Subpathway creation.....	67
6.4.2. Results.....	69
6.5. Conclusion	71
Chapter 7 The Relevance of Sequence Patterns to QPACA	72
7.1. Introduction.....	74
7.2. Methods.....	74
7.2.1. Data.....	74
7.2.2. Cluster formation	75
7.2.3. MaMF setup and evaluation.....	76
7.3. Results and discussion	78
7.3.1. Analysis of E2F binding sites in ovarian tumors.....	78
7.3.2. Permutation analysis of MaMF generated motifs in pancreatic cancer cell lines	80
Chapter 8 Conclusion and future directions	84
References.....	88
Appendix A. QPACA Model	93
A.1.1. QPACA Pathway Language.....	93
A.1.2. Sample QPACA GUI Usage.....	96
Appendix B. QPACA Analytical Module.....	98
B.1.1. PathwayAnalysis Usage	98
B.1.2. PredictGenes Usage.....	99
Appendix C. Code Documentation	100

List of Tables

Table 1. BioPAX road map.....	16
Table 2. Results table for a representative set of pathways.	53
Table 3. Breakdown of individual p-values with sample subselection for each pathway.	53
Table 4. Results for pathway recognition in subpathways.	69
Table 5. Pathway clusters.	76
Table 6. Selected MaMF motifs that are similar to E2F.....	80

List of Figures

Figure 1. Participants in transcriptional regulation.....	5
Figure 2. Overview of two-color spotted microarrays.....	8
Figure 3. QPACA overview.....	20
Figure 4. QPACA pathway representation examples.	22
Figure 5. QPACA pathway object model.	23
Figure 6. Mapping between QPACA, BioPAX and KGML object models.	25
Figure 7. QPACA optimization algorithm.....	27
Figure 8. Pseudo-code for optimization algorithm.	28
Figure 9. Magellan UML component diagram.	31
Figure 10. QPACA system architecture overview.....	32
Figure 11. Sequence diagram for QPACA visualization tools.	33
Figure 12. Sequence diagram for QPACA analytical tools.	34
Figure 13. The receptor tyrosine kinase signaling (RTK) pathway.....	37
Figure 14. The RAS subnetwork of the T-cell receptor signaling (TCR) pathway.....	38
Figure 15. Correlation in aCGH of bladder tumors.	40
Figure 16. Visually differentiating between two classes of tumors.....	43
Figure 17. Gene expression differences in cell lines treated and untreated with a MAP2K1 inhibitor.	45
Figure 18. Discriminating between treated and untreated cell lines.....	46
Figure 19. Pseudo-code for permutation methods.	52
Figure 20. Comprehensive pathway recognition results from all 191 human and yeast pathways.	55
Figure 21. Pseudo-code for augmentation algorithm.....	60
Figure 22. Comprehensive pathway augmentation results from 101 human and yeast pathways.	63
Figure 23. Pathway augmentation for the RTK pathway.	65
Figure 24. Expression heatmap of Cell Cycle pathway.....	66
Figure 25. Subpathways in the Cell Cycle pathway (hsa04110).	68

Figure 26. Comprehensive pathway augmentation results for all subpathways.....	70
Figure 27. Cell cycle pathway (hsa04110) showing clustering, subpathways, and known E2F motifs.....	78
Figure 28. Distribution of MaMF score p-values for all expression modules.....	81
Figure 29. Sample pathway description in QPACA Pathway Language.....	94
Figure 30. Analysis method selection in Magellan.....	96
Figure 31. QPACA parameter selection	96
Figure 32. QPACA data visualization results page.	97

Chapter 1

Introduction

1.1. Systems Biology

One of the major goals of current biological research is the elucidation of the complex interactions and transformations in biological systems that drive their function and behavior. This systems-based approach utilizes high-throughput techniques in the realms of functional genomics, proteomics, metabolomics and transcriptomics to generate quantitative data for the construction and validation of computational models.

While previous efforts in identifying the genes and other components in an organism have provided a valuable list, it is not sufficient for understanding the fundamental complexity of the whole system. A simple catalogue misses information about how the parts assemble to form the organism, how they interact with each other, how the organism reacts to stimuli and malfunctions, and what design principles govern the organism. The answers to these questions are important both for a greater understanding of the system as a whole as well as for enabling any modifications for improved system behavior.

A system-wide understanding of a biological organism can be broken down into four fundamental areas: system structure (assembly and interaction of the system components), system dynamics (system behavior over time), control method (system control), and design method (system design principles). Progress in all of these areas requires breakthroughs in computational science, biology, and measurement technologies, as well as integration of any discoveries with existing knowledge.

Before tackling the more complex issues of system dynamics, control, and design, it is necessary to understand the basic structure of the system. Conventionally, the study of a specific biological pathway has required painstaking research into each possible element of the network, both by conducting new series of experiments and by surveying the existing literature to identify and describe interactions. This approach has resulted in a limited, though focused, knowledge of biological networks, with hypotheses about pathway interactions that are at various levels of refinement and completeness. Computational modeling and quantitative analytical approaches seek to fill this information gap. The use of high-throughput technologies, which make it possible to survey the behavior of thousands of elements at once, allows pathway interactions to be more easily examined. Gene expression microarrays have been used extensively in an attempt to elucidate the behavior of genes based on transcriptional regulation. Post-transcriptional and post-translational information, however, have been harder to measure with high accuracy or in high throughput. (Kitano 2002)

This work focuses on this fundamental aspect of systems biology by furthering the understanding of biological pathways through the development of a generalized system for pathway visualization and analysis. While the focus on pathways clearly

requires addressing aspects of pathway definition and representation, issues that surround the data available for analysis and predictions about pathway structure are also important. These issues include the experimental methods for generating high-throughput data, as well as the particular statistical problems inherent in using these techniques. The centrality of expression data for the modeling aspects of this work also requires some discussion of transcription factors and the computational methods used to study their behavior.

1.2. Modeling Pathways

Biological pathways model the inter-relationships of nucleic acids, proteins, small molecules, and processes within a biological system. They are useful constructs in studying the structure, dynamics and control of an organism. In a disease such as cancer, which is essentially a result of disorder within the structure of the system, the study of the relationships of genes to each other is particularly relevant.

Pathway models are fairly subjective, depending on the author's focus and intended objective. While they can describe biological processes down to the molecular level, including such information as rate constants and specific molecular interactions, they can also be fairly high-level, consisting merely of a set of interconnected elements. The level of detail can reflect either the author's intended use for the pathway (high level analyses don't necessarily require detailed knowledge of rate constants) or the actual level of knowledge about a particular pathway or pathway step (many holes still exist in the understanding of complex interactions in biological systems). Additionally, pathways tend to be descriptions of the general case; not all portions of a pathway may be active in a particular cell type or under a certain set of conditions.

Broadly, several major categories of biological pathways exist: metabolic pathways, molecular interaction networks, signal transduction networks, and gene regulatory networks. Metabolic pathways describe a series of enzyme catalyzed chemical reactions that form a metabolic product, which can then be used or stored by the cell. Molecular interactions networks depict actual physical binding interactions between cellular components. Signal transduction networks refer to the set of processes by which an organism senses both external and internal environmental cues and coordinates the behavior of individual cells to support the function of the organism as a whole. Finally, gene regulatory networks delineate the interactions that govern transcription of genes into mRNA and its subsequent translation into proteins.

Each pathway category comes with its own associated vocabulary, conventions, and (in some cases) exchange standards. The following chapter will go into greater detail on existing pathway models, efforts at pathway analysis and a discussion about standards for pathway data exchange.

1.3. Transcriptional Regulation Biology

Gene regulatory networks govern the execution of the genetic instructions for both the development and the functioning of an organism. These genetic instructions are encoded in the organism's genome, which is composed of strands of DNA sequence, consisting of a four-letter alphabet. Embedded within this sequence are relatively short stretches of DNA called genes, each of which encodes the specific information necessary to manufacture a particular protein. The working units of a cell are actually these proteins, which carry out the tasks necessary for the function and survival of the organism. Genes are first transcribed into messenger RNA, which is then translated into

proteins. Each gene in an organism has a specific set of functions, which are not necessarily relevant at all time points or in all cell types, thus control machinery must be present in the cell to determine which genes are turned "on" and at which times. Gene regulatory pathways describe this cellular machinery, known as transcriptional regulation. The two primary participants in this process are proteins known as transcription factors and transcription factor binding sites, 5-15 bp long DNA sequences often located in the gene promoter region. For a gene to be expressed (that is, translated into RNA and then protein), a specific set of transcription factors is required to bind to the gene's transcription factor binding sites. Each transcription factor is capable of binding to several binding sites whose sequences can differ by several bases. The correct set of transcription factors triggers binding of RNA polymerase and initiates transcription. A transcription factor can also inhibit gene expression by blocking RNA polymerase binding. Figure 1 shows the typical structure of a genetic sequence containing the gene, with its transcription and translation start sites, as well as several different transcription factor binding sites in the promoter region, which can be located several hundreds of bases upstream of the gene itself.

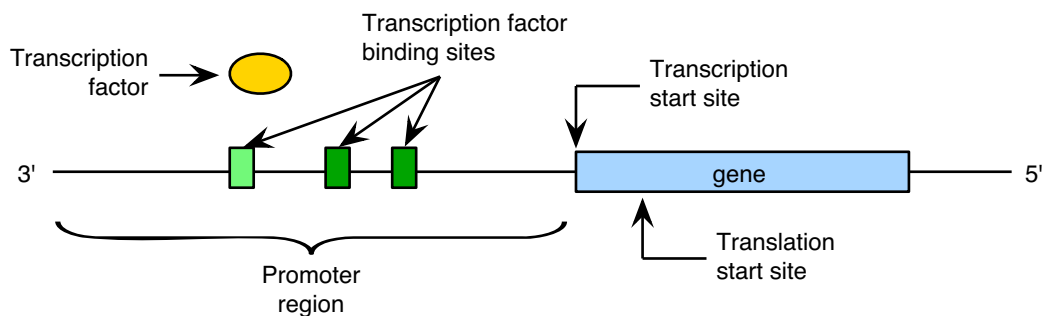


Figure 1. Participants in transcriptional regulation.

1.3.1. Motif finding

Understanding the control structure for a particular gene is dependent on knowing which transcription factor binding sites, and thus which transcription factors, are involved in driving its transcription. The process of finding these transcription factor binding sites, which are highly conserved and occur repeatedly throughout the genome, is known as motif finding. Traditional experimental methods to find binding sites for a particular transcription factor are labor intensive, resulting in many computational techniques for finding binding site sequences in promoter regions. Many different computational approaches exist in the field of motif finding, all roughly based on either pattern matching (finding new motifs based on known motifs) or pattern detection (finding over-represented novel motifs). (Elnitski, Jin et al. 2006)

Chapter 7 will discuss using MaMF, a motif finder specifically geared toward finding the types of complex transcription motifs common in higher organisms, in combination with pathway analysis to examine the transcriptional programs present within established pathways.

1.4. Microarray technologies in cancer research

This section introduces the microarray technologies, including both expression and comparative genomic hybridization arrays, used in the development and testing of this work.

Microarrays provide a powerful method for simultaneously detecting and quantifying the relative amounts of thousands nucleic acid sequences in a heterogeneous mixture. Microarrays generally work in a fairly straightforward manner. DNA sequences corresponding to the desired probe DNA are attached to a solid substrate. Then, samples

containing the target DNA are labeled with a fluorescent dye and hybridized to the array. Since binding affinities between particular probes and targets are not known, microarrays produce relative measurements of hybridization intensities.

Microarrays have generally been used to measure gene expression using cDNA derived from biological samples. Two major platforms exist for measuring gene expression: two-color spotted cDNA on glass slides and synthesized oligonucleotides on silicon. In recent years, microarrays have also been used to measure genomic aberrations in cancer. The platform used in this analysis uses a similar technology to that used in the two-color expression arrays.

1.4.1. Platforms for measuring gene expression

Two major platforms exist for measuring gene expression using microarrays: two-color hybridization onto spotted arrays and lithographically deposited oligonucleotide arrays (Affymetrix). As can be seen in Figure 2, the first platform requires comparison of two biological samples. mRNA is isolated from each sample and used for the synthesis of first strand cDNA labeled with either Cy3 (green) or Cy5 (red) fluorescent dye. Then, these labeled samples are hybridized to a glass slide containing spots of purified cDNA representing individual genes. The ratio of red to green fluorescence can be used to measure the relative concentrations of the corresponding test or reference sample for each individual mRNA represented on the slide.

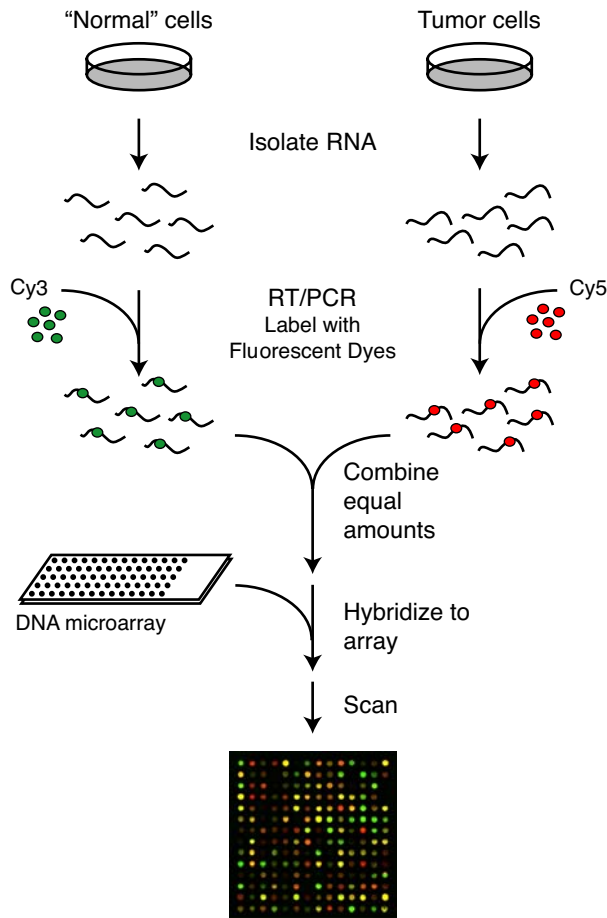


Figure 2. Overview of two-color spotted microarrays.

While this schematic shows a two-color expression microarray, the basic sequence of events is very similar for aCGH. In aCGH, instead of isolating RNA and then using RT/PCR to obtain fluorescently labeled DNA, DNA is isolated from the samples and fluorescently labeled. In the case of expression arrays, the probes on the array are gene-based DNA probes, while aCGH probes are generally a genome- or chromosome-wide set of bacterial artificial chromosomes intended to span the entire genomic space.

The Affymetrix platform, on the other hand, uses a set of ~20 bp long oligonucleotide targets, instead of cDNA targets. These targets are deposited onto a quartz slide using photolithography, a semiconductor manufacturing technique. Each gene is represented by a carefully chosen set of oligonucleotide matches and mismatches whose hybridization signals are then combined using a proprietary algorithm to form a single mRNA expression value for that gene.

Two-color arrays were the first to be widely available and are still usually cheaper to construct. They are also usually the only choice for less well-studied organisms that

don't justify commercial manufacture. Affymetrix style arrays have become more common due to generally better data quality (higher signal-to-noise ratio), better reproducibility (calculated expression values are absolute, not relative to a control sample), and decreasing costs.

1.4.2. *Microarray-based comparative genomic hybridization*

Comparative genomic hybridization (CGH) is a method for detecting copy number changes (gains, losses, or amplifications) in tumor cell genomic DNA. Traditionally, CGH has been performed by hybridizing fluorescently labeled DNA isolated from normal and cancerous tissue to a metaphase spread of normal chromosomes (FISH, or fluorescent in situ hybridization). The ratio of fluorescence intensity from these two samples indicates the gains or losses present at different physical locations in the genome. Due to the use of ratios, genomic aberrations resulting in balanced DNA content, such as tetraploidy, cannot be detected.

In array-based comparative genomic hybridization (aCGH), the samples are instead hybridized to probes printed on a glass slide, as in the two-color microarrays described above. These probes are derived both from known genes as well as from non-coding regions of the genome. The advantages of the array-based technique are improved resolution and dynamic range, as well as the ability to directly map aberrations to genomic sequence. (Pinkel, Seagraves et al. 1998; Snijders, Nowee et al. 2003)

1.5. Statistical and other issues

In an ideal world, it would be possible to find easily manipulated cell lines or organisms in which one could measure the concentration of both expressed transcripts

and translated proteins, both over time and under multiple conditions (such as knockouts, drug treatment, RNAi treatment, etc.). Given access to such data and including information about metabolite concentration and assuming that the data were at an appropriate granularity, it should be possible to work out a great deal of pathway biology. With such ideal data, many specific biochemical interactions and transformations could be identified. Data sets that approach this theoretical construct, however, are not even available in yeast and other lower organisms, much less in the human case.

The most readily available form of high-throughput data for these analyses has been microarray expression data. Even though this data is limited to the scope of transcriptional regulation as seen through mRNA expression levels, it has been a boon to research because it allows researchers to measure many different genes within the same experiment. This ability has, however, introduced problems of appropriate analysis, mainly having to do with statistical issues. Issues arising from the inherent noise in the measurement technology also limit these data. Additionally, the genes and, in large heterogeneous experiments, the samples that are actually involved in the process under study are outnumbered by those that are not involved.

1.5.1. Statistical issues of large data sets

In general, statistical measures help in determining whether or not a particular observation occurred by chance. For example, one commonly asked question in an expression microarray experiment as described above is which genes exhibit significantly highly coordinated behavior. On the surface, this question is fairly easily answered: simply calculate the correlation coefficients for all of the gene pairs and compare the results to a random permutation of the data (to assign a significance value). While this

technique would work on a smaller data set, the types of data sets encountered in microarray analysis run into something called the "multiple comparisons problem." Essentially, this problem arises whenever a large number of independent observations are evaluated using the same significance cutoff as would be used when considering one observation.

In order to understand the reason why large data sets have inherent statistical issues, it is first necessary to understand the underlying statistics involved in determining whether or not the observed data has occurred by chance.

This question falls into the statistical area of hypothesis testing, which seeks to differentiate between a null and an alternate hypothesis. The null hypothesis is defined as the theory, either believed to be true or used as a basis for argument, that there is no difference between two observations. The alternate hypothesis is that a difference does exist between the two observations. To carry out this test, a statistic is defined to measure the difference between the two observations. Then, this measurement is compared to a theoretically determined or empirically calculated distribution of that statistic under the null hypothesis. The probability value (p-value) of the hypothesis test is defined as the probability of arriving at a value of the test statistic as extreme or more extreme than those observed by chance alone. Small p-values suggest that the null hypothesis is unlikely to be true. Thus, a p-value of less than 0.05 indicates that 5% of the observed values could occur by chance.

Generally, a p-value ≤ 0.05 is considered to be significant. When considering 20 observations, one would expect to see one observation with a significant p-value purely by chance. A typical microarray experiment can contain tens of thousands of genes. An

uncorrected p-value would then result in hundreds of false positives. Several approaches have been used to calculate corrected p-values for high dimensional data sets. (Westfall and Young 1993; Jain, Chin et al. 2001; Tusher, Tibshirani et al. 2001; Olshen and Jain 2002; Segal, Dahlquist et al. 2003; Storey and Tibshirani 2003).

This work will discuss one way of avoiding the multiple comparisons problem by using orthogonal annotation and data sources to reduce the number of observations.

1.5.2. Issues with heterogeneous data sets

In the ideal case as described above, a data set used for examining a pathway would contain a carefully chosen set of perturbations that could be used to find new pathway elements and deduce the connections between them. In yeast and bacterial systems, such controlled perturbations of a system state have yielded biologically reasonable results (Friedman, Linial et al. 2000; Ideker, Thorsson et al. 2001; Paley and Karp 2002). This method relies on using prior knowledge of a pathway to generate hypotheses that can be tested using microarray data. The results can then be used to augment the pathway structure.

It has also been observed in yeast that only certain perturbations of a system are relevant to the corresponding pathways. In Hughes' yeast compendium, perturbations of genes in a particular pathway yielded the most informative changes in gene expression within that pathway, though not the largest. While deletion of *erg* genes, for example, had large effects on amino acid biosynthesis (as did many other deletions), the most specific (though smaller) effects were on the expression of genes in the *erg* pathway. (Hughes, Marton et al. 2000)

In higher organisms, it is frequently impossible or at least impractical to generate the types of controlled perturbations necessary for a systematic analysis of a pathway as described above. The data sets typically available for these organisms are fairly heterogeneous. Human cancer data sets, for example, frequently contain multiple instances of the cancer phenotype, across many individuals (and possibly multiple tumor types), each representing a unique experiment of nature. Given that each sample in such a data set has a different set of genomic, genetic, and epigenetic dysregulations, there is no reason to expect, *a priori*, that all of the samples will be relevant to a quantitative study of a particular set of genes in a particular pathway.

Chapter 3 will describe a method for selecting relevant perturbations from the type of large heterogeneous data sets frequently encountered when studying higher organisms. Chapter 5 and Chapter 6 will validate this method using both yeast and human data sets and pathways.

Chapter 2

Review of Pathway Literature

2.1. Introduction

The last chapter briefly introduced the concept of biological pathways, the different types of pathways, and the types of data generally available for pathway analysis. This chapter will focus on existing efforts in modeling, storing and exchanging pathway information; in analyzing pathway structure; and in using pathways to further analysis of large data sets.

Pathway modeling can be conceptually divided into three main sub-areas: 1) representation and curation, 2) data analysis, and 3) model recognition and augmentation.

2.2. Pathway representation and curation

As discussed in Chapter 1, many different types of pathways exist, each with its own unique set of representational challenges as well as existing representational conventions, vocabulary and in some cases even exchange standards.

In terms of representing structure, pathways are complex both in the large number of possible interactions, the many possible variations of components and connections, and the different levels of known detail for different pathways.

Pathway curation efforts are extensive and varied. Pathguide (Bader, Cary et al. 2006) lists over 200 biological pathway resources, ranging from organism specific pathway databases, such as EcoCyc (Karp, Riley et al. 2002), to multi-organism databases, such as KEGG (Kanehisa 1997; Kanehisa and Goto 2000) to pathway diagramming and analysis tools, such as GenMAPP (Dahlquist, Salomonis et al. 2002). In order for this data to be useful it needs to be accessible in a useful manner. For the casual browser, a visual representation of a pathway, such as the diagrams provided by BioCarta (<http://www.biocarta.com/genes/index.asp>), may be sufficient. In order to integrate pathways with data and perform complex analyses, however, it is necessary to extract the actual data from the databases and transform it into a representation that is computationally accessible.

One proposal for representing pathways, both as useful visual aides and as useful computational tools, is the Molecular Interaction Map (MIM). MIMs are a diagram convention that incorporate the following attributes: each molecular species should appear only once; multimolecular complexes should be represented concisely and extensibly; protein modifications should be represented uniquely, and any significant combination of molecular complexes should be represented (Kohn 1999). These attributes are important to consider when creating representations that are both rich in detail and computationally tractable.

Unfortunately, while many of the existing databases make their data accessible in computationally readable formats, each databases uses its own vocabulary and representational conventions. Frequently, two databases will have different names for the

same object or the same name for two different objects. Combining data from several different sources can be quite labor intensive.

2.2.1. Pathway exchange formats

Currently, there is no universally used standard for pathway exchange, though three formats have been proposed and are under development: PSI-MI (Proteomics Standards Initiative - Molecular Interaction), which focuses on representing protein interaction data; SBML (System Biology Markup Language), which focuses on biochemical network models, and BioPAX (Biological Pathway data exchange), which is the most general of the three and can encompass data from the other two. Table 1 shows a roadmap for the development and scope of the BioPAX format. As of this writing, BioPAX is at level 2, which allows for encoding all types of pathway elements, but is restricted to metabolic pathways and molecular interactions. Chapter 3 will further discuss the use of the BioPAX format in this work.

Table 1. BioPAX road map

Development level	Scope of format	Sample data sources
Level 1	Metabolic pathways	aMAZE, BioCyc, KEGG, PUMA2
Level 2	Level 1 plus molecular interactions	BIND, HPRD, intAct, MINT
Level 3	Level 2 plus signaling pathways and gene regulation	CSNDB, INOH, PATIKA, Reactome, TRANSPATH
Level 4	Level 3 plus genetic interactions	FlyBase, MIPS
Future levels	Level 4 plus abstract associations	PubGene, GeneWays

For a complete list of the databases currently using the BioPAX format, see www.pathguide.org.

2.3. Pathway-based data analysis

Pathway-based data analysis can be either qualitative (using a pathway diagram to map out data values) or quantitative (combining pathway-based information with data analysis to reach a quantitative result). Many of the existing pathway databases allow the former type of analysis. Separate pathway analysis tools, such as Cytoscape (Shannon,

Markiel et al. 2003), also exist for this type of analysis as well as for more complex qualitative and quantitative diagram-centered analyses.

Pathway-based quantitative data analysis, such as the Barkai ‘transcription module’ research (Ihmels, Friedlander et al. 2002), is an example of pathway-based quantitative data analysis. A transcription module is the set of genes and experimental conditions (data samples) that contribute to their co-regulation. They find these transcription modules by beginning with a set of genes suspected to belong to a transcription module (based on annotation or sequence similarity) and then using an algorithm to first choose relevant experiments and then refine the gene set to produce a common signature that can be used to identify these transcription modules. Similarly, Stuart *et. al.* (Stuart, Segal et al. 2003) showed that, by combining co-expression relationships found in many different experiments in several different organisms, it is possible to deduce functional relationships between genes.

2.4. Model recognition and induction

Attempts to induce pathway structure have remained separate from the efforts to represent already existing knowledge. While some of the representation efforts, like KEGG (Nakao, Bono et al. 1999), have also delved into the pathway prediction field, many have approached this problem from a different direction. Pathway representation and data visualization have been seen as ways of gathering knowledge for use by the scientific community, but pathway prediction efforts have focused on mathematical approaches to extracting interaction information from raw data.

Previous efforts to predict biological networks from expression data have relied on three major types of algorithms: pair-wise, equation-based, and network-based. While

each one has yielded important results, all of them have specific limitations. Pair-wise algorithms are based on correlations (Arkin, Shen et al. 1997) and are thus unable to determine causation. Equation-based algorithms, which can be linear (D'Haeseleer, Wen et al. 1999), non-linear (Weaver, Workman et al. 1999), or differential (Arkin, Ross et al. 1998), are either very coarse or require detailed knowledge of biochemical interactions. Finally, network-based algorithms are either Boolean (Liang, Fuhrman et al. 1998; Akutsu, Miyano et al. 2000), which requires discretization of data and thus loss of information, or Bayesian (Friedman, Linial et al. 2000; Hartemink, Gifford et al. 2001; Jansen, Yu et al. 2003), which can be biased toward more simple models in the absence of enough data (Smith, Jarvis et al. 2003). Many of these algorithms require making assumptions that are known to be untrue (e.g. that biological networks contain no feedback cycles). The validity of these different methods tends to be difficult to assess because many of the interactions have not been biologically tested, nor are common benchmarks yet available in this relatively young field. With the advent of comprehensive yeast expression datasets, some labs (Smith, Jarvis et al. 2003) have begun to try more data-driven approaches that rely less on mathematical formalisms.

2.5. Conclusion

The work differs from the aforementioned efforts in three ways. First, on a representational level, QPACA explicitly represents the objects and inter-relationships that comprise a pathway. The visual depiction is automatically generated. This is precisely the converse of efforts such as KEGG and BioCarta, where the diagrams are the primary representation and the pathway must be derived from the depiction. Second, while many curation-oriented efforts focus on comprehensive but static pathway

descriptions over a broad set of processes, this effort focuses on constructing tools that allow domain experts to rapidly construct and modify pathways for use in data exploration, which may itself feed back onto the original pathway structure. Third, many of the pathway recognition, induction, and network modeling approaches are rooted in formalisms that either restrict representational richness (no feedback cycles, etc.) or require substantial data reduction (binary or ternary states) that may limit their utility. QPACA's framework imposes no such limitations and introduces novel methodological development to address problems of pathway recognition and augmentation.

Chapter 3

QPACA: System Design

3.1. Introduction

The previous chapters demonstrate both the necessity for pathway-based analytical tools as well as the challenges inherent in the creation of these tools. QPACA, **Q**uantitative **P**athway **A**nalysis in **C**ancer, was developed with the goal of creating an easy to use system for the exploration (both visual and computational) of both fine- and coarse-grained pathway structure and of experimental data within the context of that structure. At the core of this system lies a simple and flexible pathway model. Using this model, it is possible to interrogate relationships between genes within a pathway as well as to visualize pathways and data. More importantly, QPACA also incorporates a set of analytical tools for pathway recognition and augmentation.

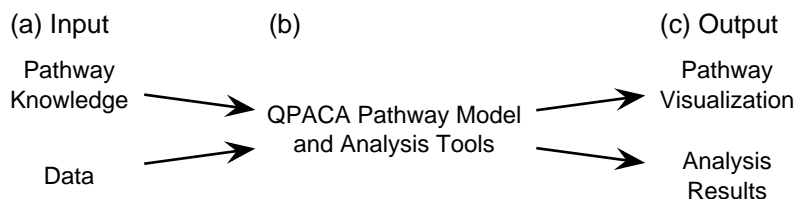


Figure 3. QPACA overview.

The QPACA system (b) accepts inputs in the form of pathway knowledge (BioPAX, KGML, or QPACA formatted text files) and data (tab-delimited text files). The system presents outputs as either a visual representation of the pathway input or as computational analysis results.

As can be seen in Figure 3, QPACA takes well-defined input (both pathways and data), feeds it through the pathway model and analysis tools, and then outputs pathway visualizations based on those inputs and a set of analysis parameters. This chapter begins with a discussion of the QPACA pathway model and how external pathway knowledge is translated and utilized by the system. Then, it discusses the optimization algorithm that forms the basis of QPACA's analytical tools. It concludes with a discussion of how these pieces fit together within the system architecture.

Application of the system to the exploration of three biological datasets will be discussed in Chapter 4, while Chapter 5 and Chapter 6 will discuss the development and testing of the QPACA analytical tools.

3.2. Pathway model

The development of pathway analysis tools was dependent on a solid pathway model that was ideally both intuitive and flexible. Specifically, QPACA's pathway model needed to be accessible to the average biologist as well as flexible enough to handle the myriad different types of pathways that exist in biology. At the time it was developed, no such pathway model existed.

QPACA models pathways as directed graphs, a computational structure composed of nodes connected by edges. Directed graphs offer the benefit of an existing body of algorithms for navigation, visualization and structure analysis. Within this structure, QPACA models physical entities and processes in pathways as nodes. The various types of interactions between these entities and processes are then modeled as edges connecting the nodes. To encompass the complexity inherent in pathway structure without losing the flexibility of the directed graph structure, QPACA's model also includes a special type of

node called the "event" node. Rather than representing a concrete concept in the pathway, this node type represents an event that occurs in the pathway, such as phosphorylation or the formation of protein complexes. In addition, each of these pathway components can be assigned any number of attributes stored as user-defined name-value pairs. Several different types of pathway interactions mapped to the QPACA representation can be seen in Figure 4.

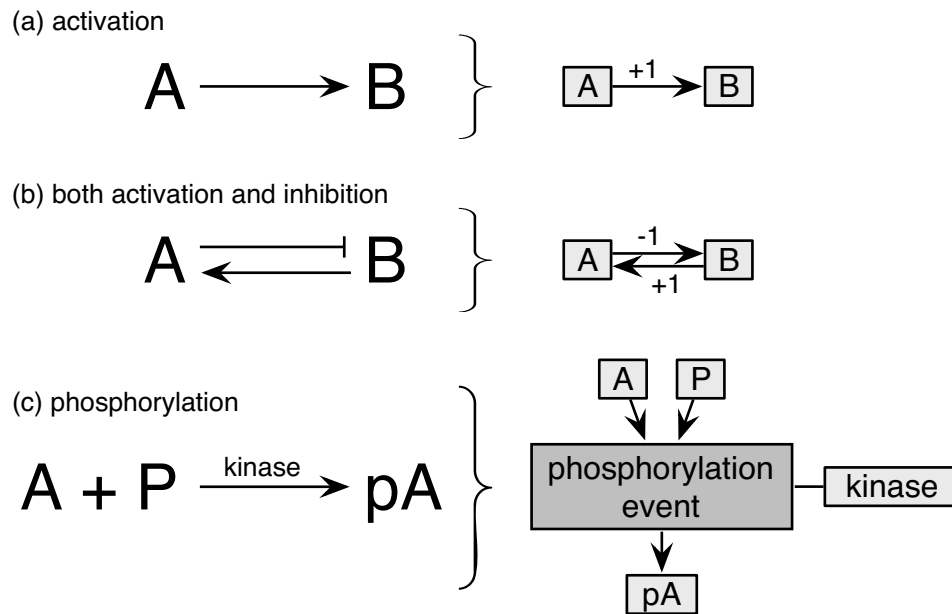


Figure 4. QPACA pathway representation examples.

On the left are several typical pathway interactions. On the right are those interactions mapped to QPACA pathway components. QPACA nodes are indicated as boxes, while the edges are denoted as lines connecting the boxes. The direction of the arrows indicates the direction of the interaction. (a) and (b) show typical signaling interactions with no modifying event. QPACA stores this information as two nodes (representing each side of the signaling interaction) connected by an edge that has either a positive (activation) or negative (inhibition) edge type attribute. Note that, as in (b), nodes may have multiple edges connecting them, signifying different possible interactions, reversible reactions, etc. A more complex interaction (such as phosphorylation in (c)) is represented using an event node. The edges in this case do not have edge type modifiers. Any physical entities involved in aiding the interaction are connected to the event node with an undirected edge.

One further refinement that QPACA makes to the basic directed graph structure is the notion of assemblies of pathway components. In Figure 4c, two pathway components, A and phosphate, join together to form phosphorylated A. QPACA is capable of representing this "assembly" of two items as a separate component composed of the two

original components. Assemblies are essentially any collection of pathway components that acts as one node in an interaction. This concept encompasses both the idea of physically assembled groups of objects, such as protein complexes, as well as interchangeable groups of objects, such as families of proteins that may act interchangeably at a particular node in the pathway structure.

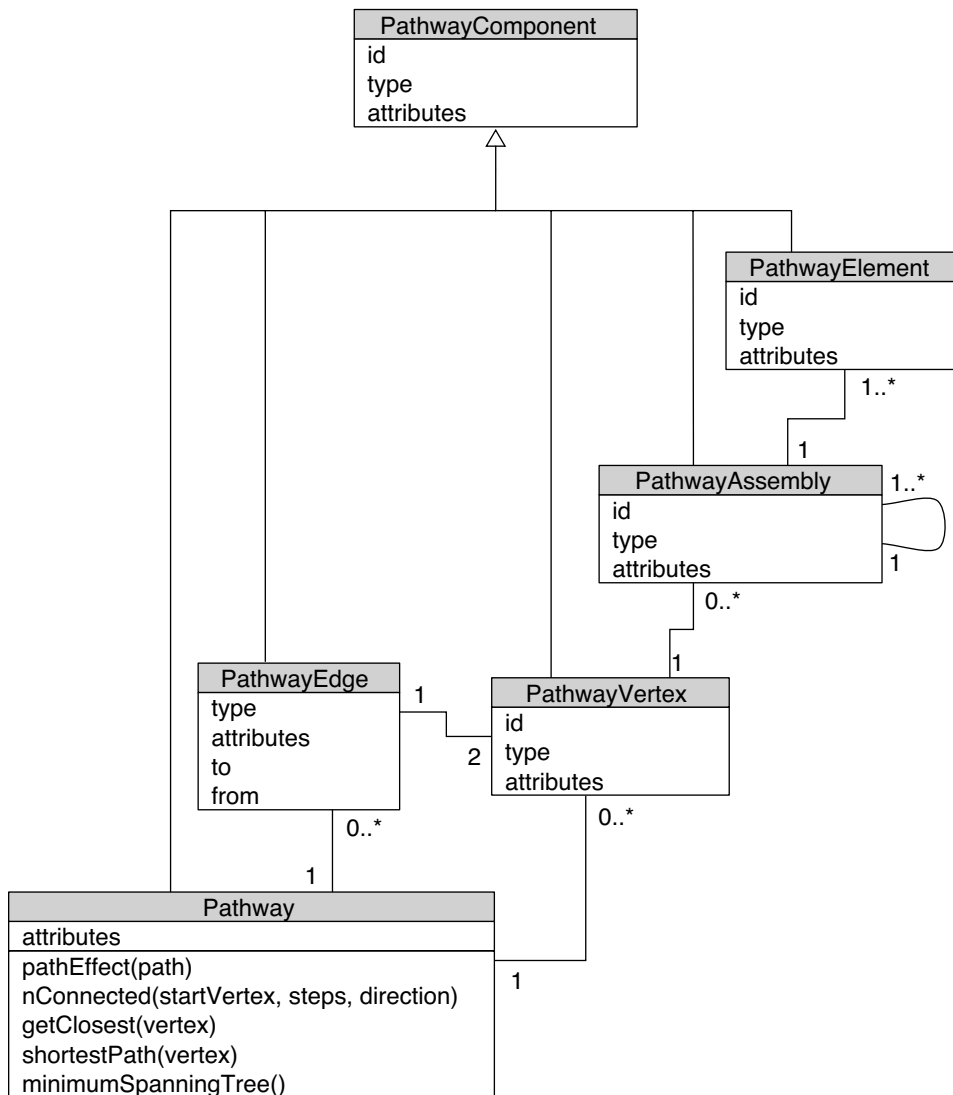


Figure 5. QPACA pathway object model.

Figure 5 shows the pathway object model that corresponds to the QPACA pathway representation. This figure details the composition of each pathway object and

their relationships with each other. Each object in the model inherits from the PathwayComponent object, which ensures that each child object has an id, type and set of attributes. The id and type of each object serve to uniquely identify it and provide some basic information about its nature. The model supports arbitrary ids and types, though pre-defined types are necessary for accurate visualization and navigation. Attributes are a set of name-value pairs that store user-defined sets of annotations for each object. The smallest unit of the pathway is the PathwayElement, which represents a single physical element (such as a protein or molecule) or process within a pathway. PathwayAssemblies are containers of PathwayElements or other PathwayAssemblies that represent complexes or families of pathway elements. The PathwayEdge and PathwayVertex objects encode all information about the pathway relationships. PathwayVertices can be either simple nodes, which must contain a PathwayAssembly representing the object found at that node, or event nodes, which may contain a PathwayAssembly indicating an object that facilitates the event (such as a catalyst). PathwayEdges encode the relationships between different nodes. Finally, the Pathway object contains the PathwayVertices and PathwayEdges and provides methods for querying the pathway objects and navigating the pathway structure.

QPACA includes a set of filters for accessing pathway information from several different file formats: KEGG Markup Language (KGML)(Kanehisa, Goto et al. 2004), BioPAX (<http://www.biopax.org>) and the QPACA pathway language (see A.1.1). The explicit mapping between the different object models can be seen in Figure 6. The QPACA language allows biologists to directly enter their pathways into QPACA, and support for both BioPAX and KGML allow QPACA to be used with the existing pathway

databases. While BioPAX is being adopted as a pathway exchange standard by most of the public pathway databases (such as Reactome and KEGG) as well as by caBIG, at this writing it is still at level 2. Signaling pathways and gene regulation networks will not be fully supported until level 3. In order to facilitate use with all of the available KEGG pathways, the KGML filter was also added.

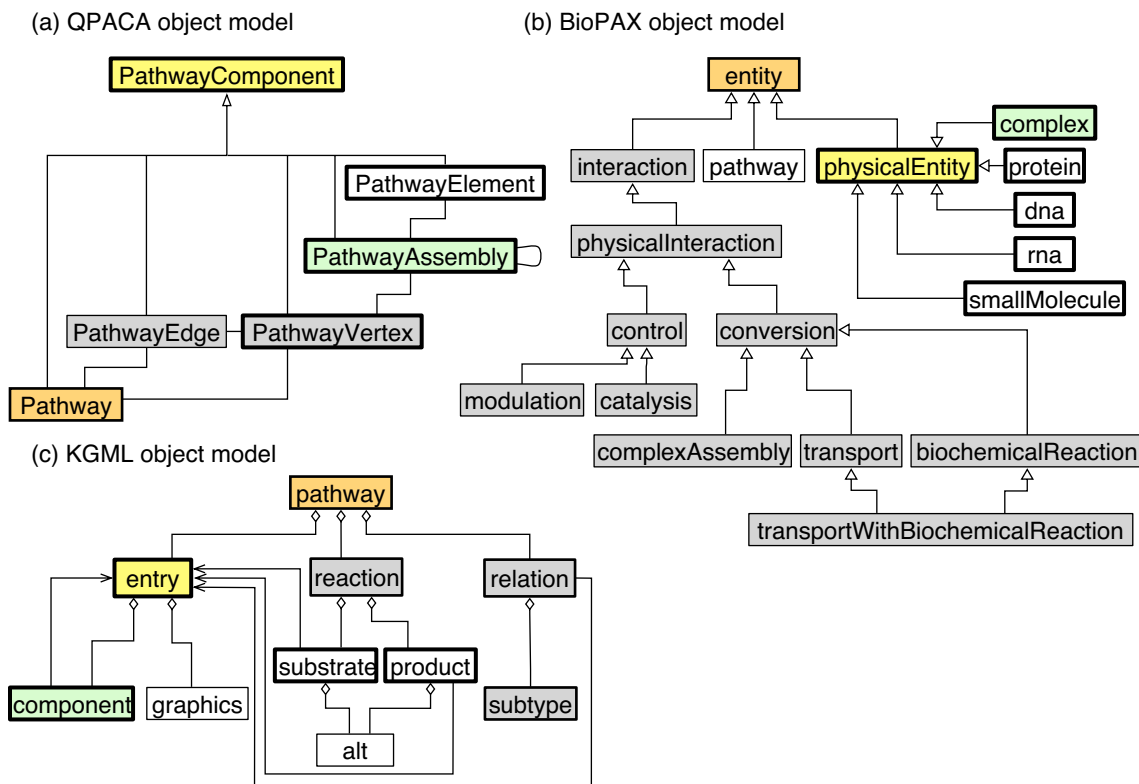


Figure 6. Mapping between QPACA, BioPAX and KGML object models.

The mappings between the three different object models are indicated in color and are fairly straightforward. All of these pathway models are based on the directed graphs with nodes and edges. Nodes and their corresponding physical entities are outlined in black. Edges, representing the different types of possible interactions are shaded in gray. Each model explicitly represents a pathway object, shaded in orange. Each model also has a parent object that represents a generic physical object (shaded in yellow) as well as an object that represents complex groups (shaded in green).

3.3. QPACA optimization method

Once a pathway model is in place, it can be used to provide support for pathway based analytical analyses. QPACA's analytical methods take a set of pathway genes and

employ an optimization method to filter the data set such that it concentrates data that are most informative for that particular set of genes.

Chapter 1 (Section 1.5.2) discussed the specific example of the erg pathway in the Hughes yeast compendium, which showed that those samples with perturbations of genes in a particular pathway yielded the most informative changes in gene expression in that pathway. Similarly, given the general heterogeneity of the typical human cancer data set, it is unlikely that all samples in a given data set will be informative for a particular set of selected genes. QPACA resolves this issue by automatically selecting these relevant samples through the maximization of a scoring metric (see Figure 7). The goal of this method is to identify a specific subset of biological samples where the variation of gene expression across these samples conveys information about the function of the genes being studied. In simpler terms, this method can be used as a tool for pathway recognition, the ability to discriminate between real pathways and non-pathways.

A specific discussion of QPACA's ability to distinguish pathways from non-pathways using microarray expression data, given a set of genes hypothesized to be part of a pathway or coordinated process, can be found in Chapter 5.

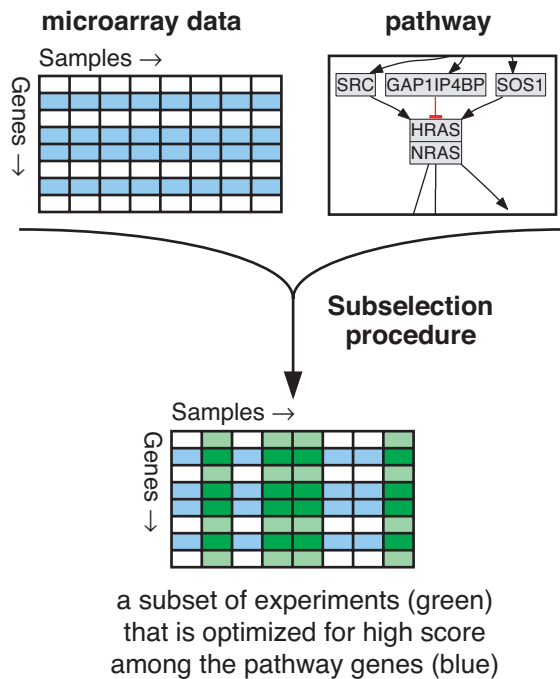


Figure 7. QPACA optimization algorithm.

Sample subselection is achieved through the use of an optimization method that takes as input a large microarray data set, a set of genes, and a number of iterations. The method follows a simple hill-climbing optimization approach. Initially, a random subset of samples is chosen and scored. Then, in each subsequent iteration, a random sample is swapped into the subset and a new score is calculated. If the new score is better than the old score, the new subset is retained for the next iteration; otherwise the subset reverts to its previous state. The scoring metric was the median of the gene-gene correlations across all pairs of genes within a sample subset.

The optimization method employed in sample subselection follows a simple hill-climbing approach. A data set consists of a matrix of log-expression changes (denoted E) of a gene $g \in G\{1, \dots, N\}$ over experimental samples $s \in S\{1, \dots, M\}$, where N and M are the number of genes and samples, respectively. The optimization finds the optimal sample subset, S_{path} , for a specific subset of genes, G_{path} . To decide whether a set of genes is part of a pathway, G_{path} is composed of all genes in the data set that belong to that specific putative pathway. Figure 8 shows a pseudo-code representation of the optimization procedure.

```

procedure optimize( $G_{\text{path}}, E$ )
for  $i$  in (1:numStartPoints)
    randomly select subset of samples  $S_{\text{init}} \subset S$ 
    numIter := 0
     $S_{\text{current}} := S_{\text{init}}$ 
    do until (no change in  $S_{\text{current}}$ )
        or (numIter == maxIter)
             $s_{\text{new}} :=$  pick random sample  $s_{\text{new}} \in S \cap S_{\text{current}}$ 
             $s_{\text{old}} :=$  pick random sample  $s_{\text{old}} \in S_{\text{current}}$ 
             $S_{\text{test}} := (S_{\text{current}} - s_{\text{old}}) \cup s_{\text{new}}$ 
            if ( $\text{score}(G_{\text{path}}, S_{\text{test}}, E) > \text{score}(G_{\text{path}}, S_{\text{current}}, E)$ )
                 $S_{\text{current}} := S_{\text{test}}$ 
            ++numIter
    if (numIter == 0) or ( $\text{score}(G_{\text{path}}, S_{\text{current}}, E) > \text{score}(G_{\text{path}}, S_{\text{path}}, E)$ )
         $S_{\text{path}} := S_{\text{current}}$ 
end

```

Figure 8. Pseudo-code for optimization algorithm.

$\text{score}(\text{gene subset, sample subset, expression matrix})$ is defined as the median of all gene-gene correlations (using Pearson's correlation) across all pairs of genes within the gene subset over the sample subset given the expression matrix. In the described experiments, maxIter was set to 600 and numStartPoints was set to 20 based on empirical evidence that these numbers ensure convergence.

One application of the method is to test whether a gene set is behaving in a coordinated fashion, and whether the coordination is relevant to our notion of a biological pathway. This test pertains to the common biological observation that some small set of genes appears to be behaving similarly (as in a hierarchical clustering of expression patterns), but where it is important to ask more quantitatively whether this gene set is also related through a common pathway. Chapter 5 reports on the algorithm's ability to yield high scores on sets of known pathway genes as compared with random sets of genes (to simulate false pathways) or randomized expression data (to simulate data with no signal).

The optimization method may also be applied to the question of biological pathway augmentation. Given a pathway for which the algorithm was able to recognize the gene set as a pathway, the scoring metric can then be used to rank the remaining non-pathway genes. Highly ranked genes could be hypothesized to also belong to the pathway. Chapter 6 reports on the algorithm's ability to find new pathway genes by

conducting a series of cross-validation experiments in which a percentage of randomly chosen known pathway genes were held out during optimization and scoring of the pathway. Then, QPACA's scoring methodology was used to rank-order the held-out pathway genes and compare them to the background of non-pathway genes: those in other known pathways, those not known to belong to any other pathways and those in other closely related pathways.

However, ranking genes using the scoring metric described in Figure 8 (the median of all gene-gene correlations) results in many tied scores. As a result, Chapter 6 will also show the use of a refined scoring metric that substitutes the average of the top 10% of all gene-gene correlations, which reduces the number of tied genes.

3.4. System architecture

3.4.1. *Design overview*

At a basic level, QPACA can be seen as a black box that takes the pathway information and data that are input into the system and outputs a set of annotated images or data based on a set of user specified parameters (see Figure 3). This black box contains three basic modules: the pathway model (which controls input and output of pathway information), the visualization tools (which create annotated images) and the analysis tools (which generate pathway recognition and prediction results). The pathway model and associated code is responsible for input and output of pathway information. The visual and analysis tools both access the pathway model and share a common user interface, but are otherwise separate toolsets. Each of the modules can be accessed independently from the command-line.

The full system is organized in a 3-tier architecture (see Figure 10). This structure allows for the logical separation of user-facing client, the system's application logic, and the back-end database and tools. The user sees a set of dynamically generated web pages within a simple web-browser, removing the need for separate installation of QPACA for each user. These pages are dynamically generated by the application server based on inputs received from the user and information retrieved from the database and generated by the QPACA tools.

3.4.2. *Implementation*

3.4.2.1. *Magellan*

Magellan is a web-based system for the upload, storage, and analysis of multivariate data as well as annotations. It allows biologists to perform analyses of their data over the Internet through the use of a server-based system framework that interfaces with many different analytical systems. To maximize these different types of information that the system can handle, data and annotations are treated as abstract entities with a user-defined type and value. Annotations may be used for analyses or visualization, as well as for intelligent subsetting of larger datasets. Analytical methods are deployed in a modular fashion that allows straightforward additions of new functionality or changes to existing functionality. Figure 9 diagrams the different components of the system and their interaction with each other.

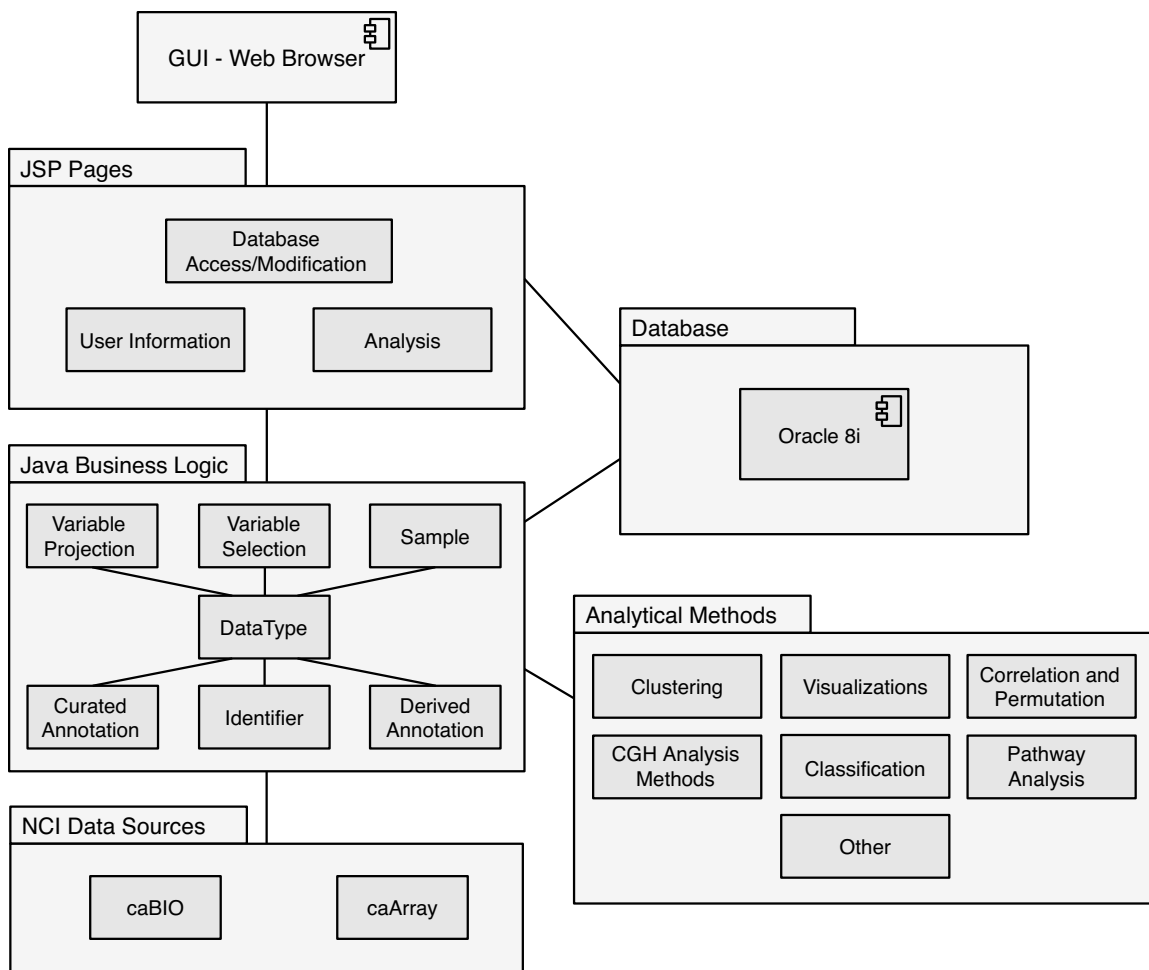


Figure 9. Magellan UML component diagram.

This diagram shows the various interactions of the different components of Magellan. Magellan is roughly divided into a graphical user interface (GUI), a database, Java-based business logic, Java Server Pages (JSP) presentation pages, analytical methods, and NCI data sources. The JSP pages present information from the database and the analytical methods to the user via the web browser GUI. They directly contact the database for accessing user information as well as general data access and modification. The Java business logic is responsible for coordinating all interaction with the data for analyses as well as coordinating access to the outside NCI data sources. Analytical methods exist as separate modules, allowing straightforward modification of existing methods as well as addition of new methods.

3.4.2.2. QPACA as an analysis module

Figure 10 shows the structure of QPACA and its relationship to the different components of the Magellan system. As a part of Magellan, the QPACA toolset is available to researchers through a standard web browser. The specific interfaces between the QPACA tools and Magellan are detailed in the sequence diagrams in Figure 11 and Figure 12. Additionally, the QPACA tools can also be directly installed on the client

machine as stand-alone command-line programs. See Appendix B for further documentation.

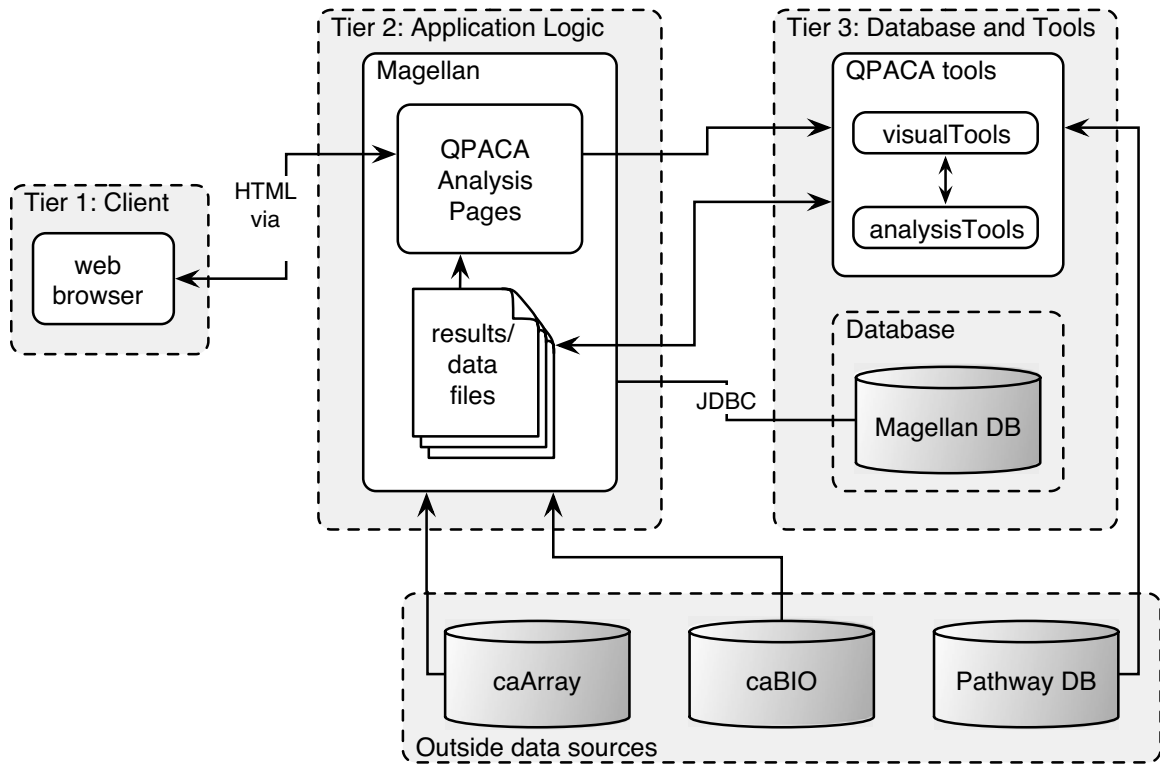


Figure 10. QPACA system architecture overview.

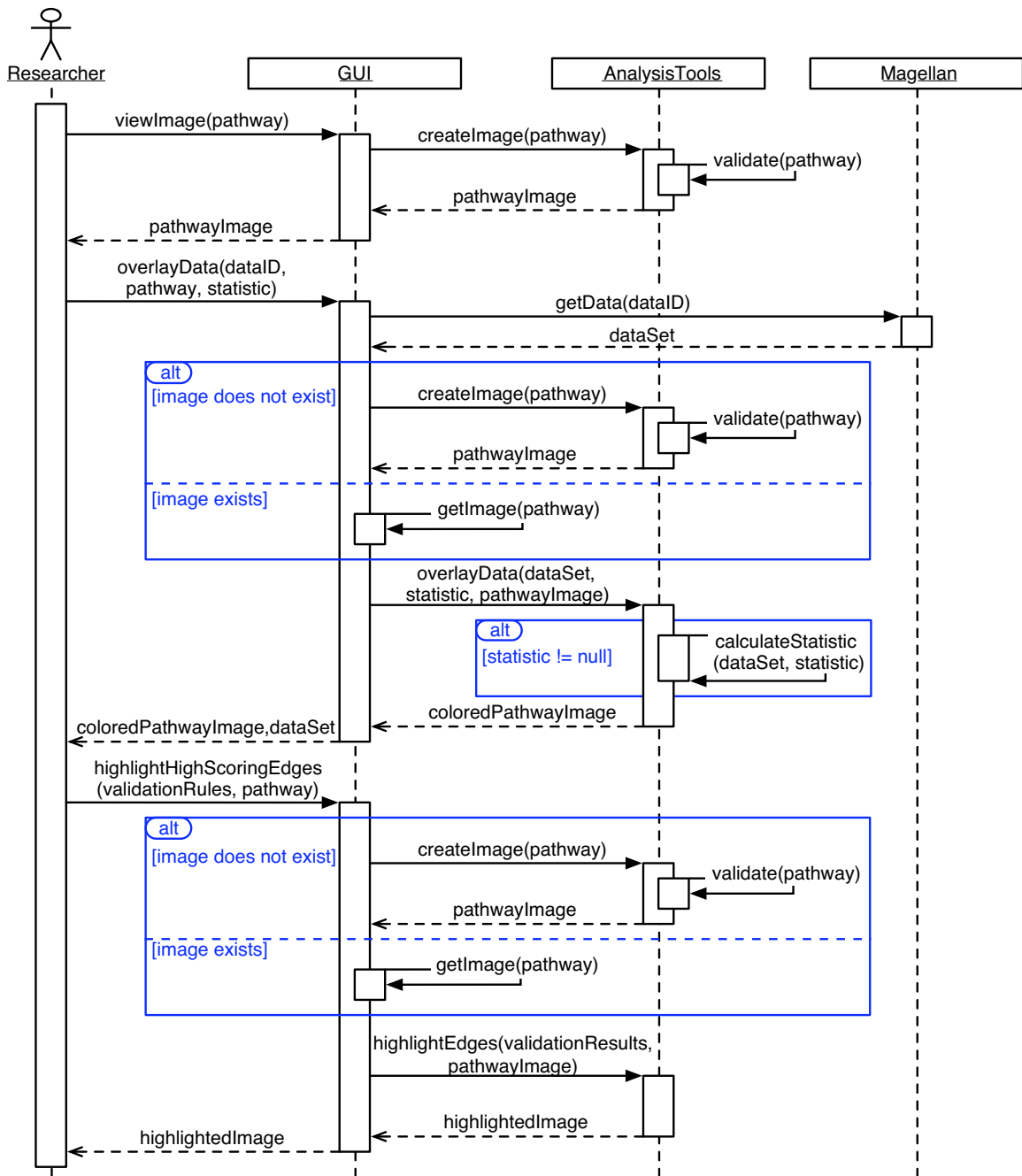


Figure 11. Sequence diagram for QPACA visualization tools.

Interactions between the QPACA visualization tools and the Magellan GUI, Magellan database, and user are indicated as a series of lines denoting actions and boxes indicating duration. The blue boxes represent different possible actions based on pre-existing conditions. The actions shown encompass the different possible ways to view images within the QPACA system: (1) simply viewing the automatically generated image, (2) coloring an image based on a specified dataset or computed statistic, and (3) coloring an image based on pathway recognition results from the Analysis Tools (see Figure 12).

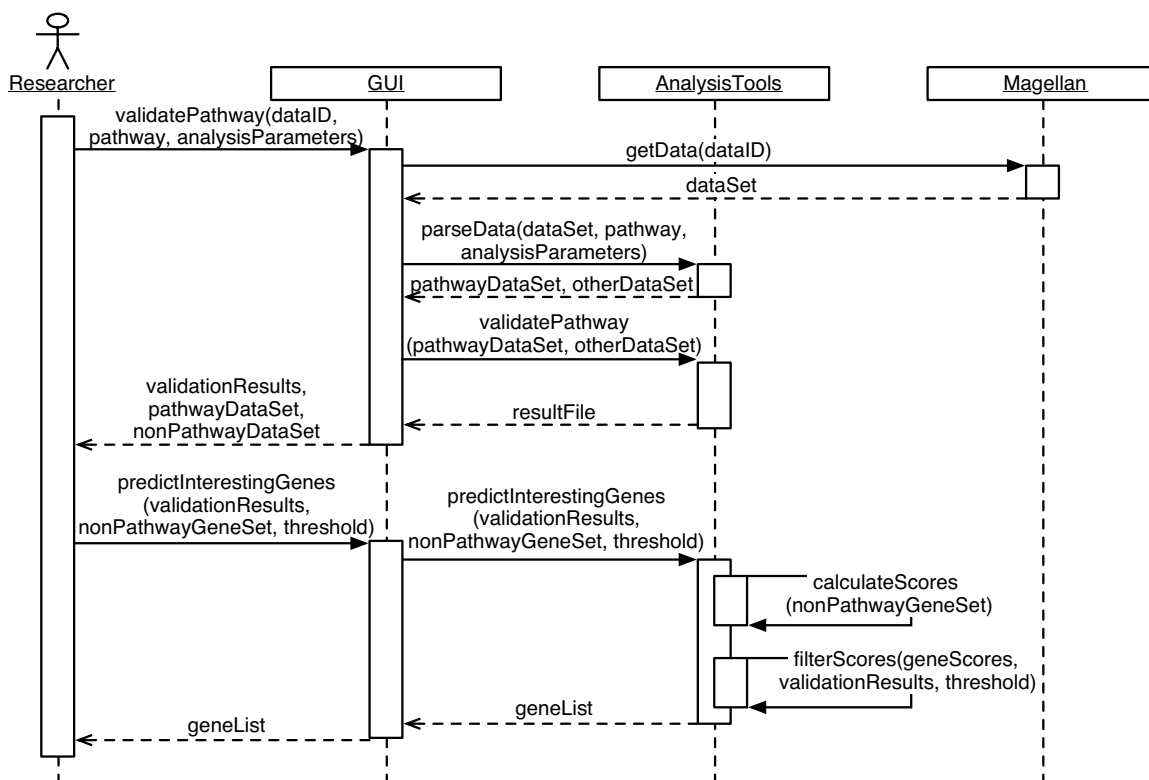


Figure 12. Sequence diagram for QPACA analytical tools.

Interactions between the QPACA analysis tools and the Magellan GUI, Magellan database, and user are indicated as a series of lines denoting actions and boxes indicating duration. Two major actions are shown: (1) `validatePathway`, which performs the pathway recognition analysis task, and (2) `predictInterestingGenes`, which performs the pathway prediction analysis task.

3.4.3. QPACA toolset design

The QPACA toolset is composed of three separate modules: the pathway model, the visualization tools, and the analysis tools. The first two are written in Java, while the last is written in C for greater computational speed. The module containing the pathway model can be used to control input and output of pathway information as well as to navigate the pathway structure and answer basic questions based purely on the structure using standard graph-based algorithms. The model is built on the OpenJGraph open source Java library, which provides most of the basic graph functionality. Automatically generated visual graph output is provided using Graphviz (<http://www.graphviz.org>), open source graph visualization software. As discussed in Section 3.2, pathway input can

come in three different file formats: BioPAX, KGML, or QPACA. QPACA's visualization module is also written in Java and provides classes and methods for coloring pathways based on data. This module can also calculate correlation and the F-statistic and color the pathway based on these computed values. Finally, the analysis module computes pathway recognition and augmentation results based on a dataset and gene lists derived from pathways. For further documentation of these software modules, see Appendix B.

3.5. Conclusion

This chapter introduced QPACA's system design, covering both the general architecture as well as specifics of both the pathway model and the analytical algorithms. QPACA is composed of three distinct modules: the pathway model, the visualization toolset, and the analysis toolset. The pathway model provides the basis for both the qualitative and quantitative analysis techniques. Chapter 4 will present several examples of using the visualization tools, while Chapter 5 and Chapter 6 will discuss the use and validation of the pathway recognition and pathway augmentation algorithms. Chapter 8 will explore combining these pathway analysis tools with analysis of transcription factor binding sites.

Chapter 4

Pathway Exploration and Visualization

4.1. Introduction

The previous chapter covered the design and implementation of QPACA. A working pathway model not only formalizes the representation of biological pathways, it also allows mapping from the experimental data space to the pathway space and enables researchers to ask questions about their data involving pathway structure and pathway-derived gene sets. One basic use for a pathway representation in data analysis is to simply reduce the dimensionality of a large data set to only those genes known to be involved in a known affected process, thereby avoiding the multiple comparisons problem outlined in Chapter 1. Additionally, a pathway representation can be used to provide a quick overview of the samples in a data set, creating a simple visual profile that can reveal differences in the types of samples. Finally, the gene/gene relationships of genes known to be closely related based on pathway structure can be easily queried and compared to mutational status or experimental conditions.

This chapter discusses three examples of using QPACA to explore and visualize data in the context of pathways that yield interesting insights into gene/gene relationships

RPS6KB1, PDK1, PTEN, PIK3CB, PIK3CD, PIK3R1, PIK3R2, PIK3CA, RACGAP1, PLCG1, DAG1, PKC, AP1M2, FOSB, RASGRP1, and RAP1GDS1.

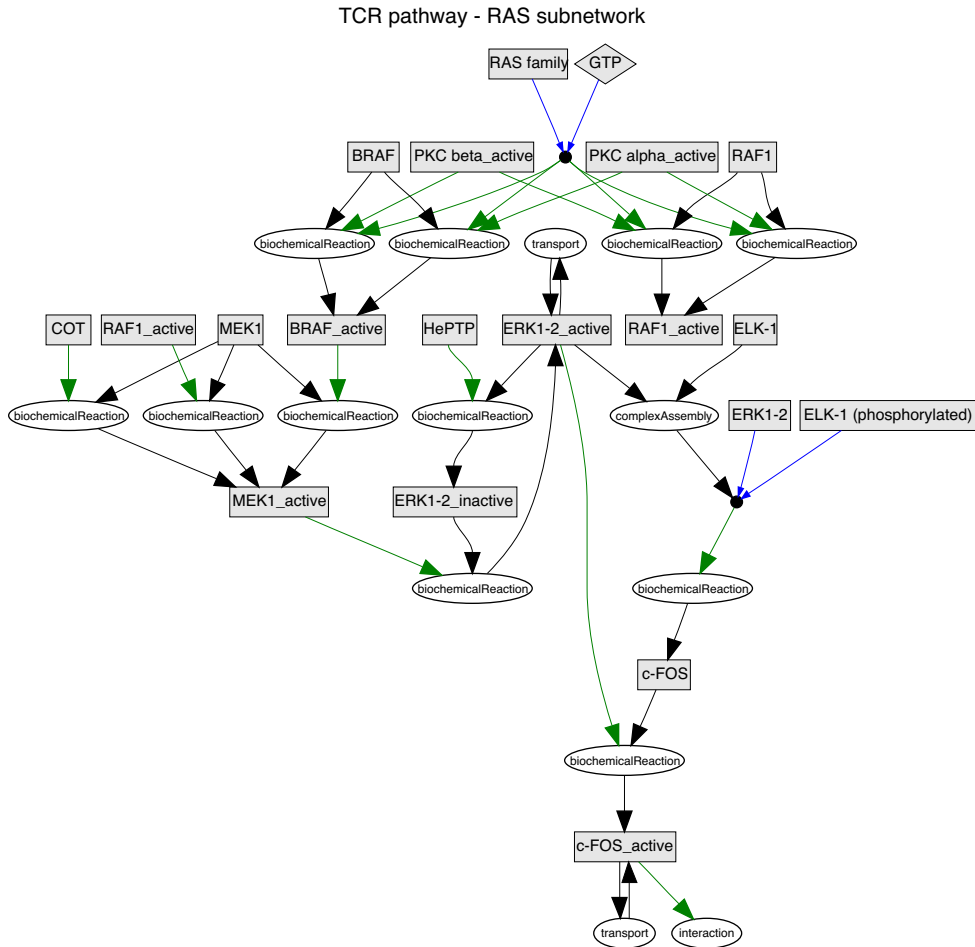


Figure 14. The RAS subnetwork of the T-cell receptor signaling (TCR) pathway.

The image was automatically generated by QPACA based on a BioPAX level 2 formatted file downloaded from NCI-Nature Pathway Interaction Database (PID) (<http://pid.nci.nih.gov>). This diagram shows two pathway features not seen in Figure 13: 1) color-coding of activation/inhibition edges (green arrows indicate activation, red bars indicate inhibition (not seen)) and 2) use of event nodes as described in section 3.x (ovals). BioPAX level 2 is not able to encode certain commonly used pathway conventions. In this case, families of proteins are not encoded in the BioPAX output file, even though they are stored in the PID. Also, interactions with other pathways cannot be easily modeled, hence the presence of a generic "interaction" event node at the bottom of the diagram, which the PID lists as a link to the Calcium subnetwork of the TCR pathway. The RAS subnetwork consists of the following 11 genes: *BRAF*, *PRKCB* (*PKC beta*), *PRKCA* (*PKC alpha*), *RAF1*, *MAP2K1* (*MEK1*), *MAP3K8* (*COT*), *MAPK3* (*ERK1*), *MAPK1* (*ERK2*), *ELK1*, *FOS* (*c-FOS*).

4.2. Exploration of gene/gene relationships in an array-based comparative genomic hybridization experiment

Statistical analysis of microarray data can be aided by utilizing the context of a known pathway. In this instance, a set of array-based comparative genomic hybridizations (aCGH) derived from primary human bladder tumors were analyzed in the context of the Receptor Tyrosine Kinase (RTK) pathway (Figure 13). CGH measures the relative copy number of tumor DNA versus normal DNA. The data set consisted of 41 bladder tumors of different stages (Veltman, Fridlyand et al. 2003). Data was obtained from both high resolution (2000 clone) and oncogene focused (500 clone) arrays of bacterial artificial chromosomes (BACs). Using the pathway representation and focusing on gene/gene relationships in S-phase checkpoint control, several subsets of candidate genes for this analysis were formed. First, all genes within six steps upstream of S-phase checkpoint control (set 1) were found via graph traversal of the RTK pathway representation. Then two subsets were formed: a) all gene pairs within this set that are within three steps of each other or which target, or are targeted by, the same gene (set 2); b) all genes that are frequently changed in the data set (set 3). The first two subsets focused on genes pairs that were believed to be closely tied to a process known to be important in tumor growth and to consist of genes that were closely related to each other. Since knowledge of pathway membership is imperfect, the third set attempted to incorporate genes that might also be involved.

Two separate statistical analyses were performed: cross-locus correlation as well as complementation and concordance. In CGH data, closely mapping loci are expected to correlate in copy number, since large deletions and amplifications spanning many loci are

common. Strong correlations between loci that map far apart, however, can indicate non-trivial interactions. For this test, the correlations for all genes in sets 1 and 3 were computed. Significance was tested by permutation. Several pairs of distantly located genes were found to be highly correlated: gain of ERBB2 (17q12) and gain of CCNE1 (19q13.11) ($p < 0.05$), loss of TP53 (17p13.3) and gain of CCND1 (11q13) ($p < 0.1$), and gain of AIB1 (20q12) and loss of PTEN (10q23) ($p < 0.05$). The last pair is particularly interesting because it incorporates a gene that was not included in the pathway description.

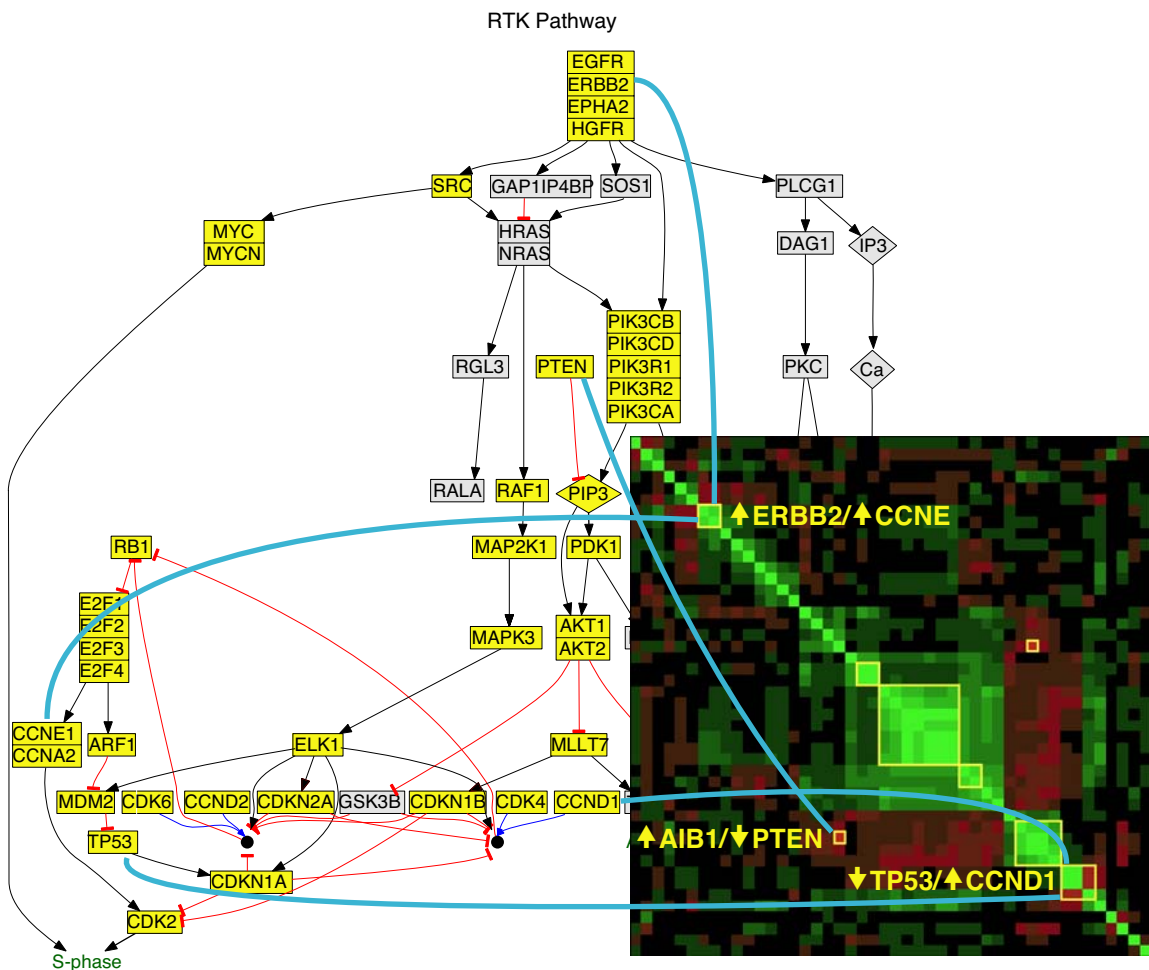


Figure 15. Correlation in aCGH of bladder tumors.

The genes highlighted in yellow in the pathway diagram represent all genes in set 1, that is, genes at least 6 steps upstream from S-phase checkpoint control. These genes, along with genes found to be frequently changed, were used in the correlation analysis. Highlighted in yellow in the red-green correlation plot on

the right are all correlations that were found to be statistically significant. Of special note are the correlations connected to the pathway diagram in blue. These are: gain of ERBB2 (17q12) and gain of CCNE1 (19q13.11) ($p < 0.05$); gain of AIB1 (20q12) and loss of PTEN (10q23) ($p < 0.05$); and loss of TP53 (17p13.3) and gain of CCND1 (11q13) ($p < 0.1$). Of these, AIB1 was included in the analysis because it was frequently changed, not due to pathway membership.

In addition to correlation, complementation and concordance were also considered. Complementation is defined as change in one gene or locus occurring instead of change in another gene or locus. Concordance is the opposite, change in one gene or locus occurring along with change in another gene or locus. For this test, the test pairs (sets 2 and 3) were compared to a set of random pairs from a null population consisting of pairs of genes that change frequently and are not located on the same chromosome. Several complementary relationships were found to be significant: gain of CCND1 behaved complementarily with gain of both E2F3 ($p < 0.05$) and CCNE1 ($p < 0.05$); loss of CDKN2A was complementary to gain in CCND1 ($p < 0.1$).

When examined in the context of the RTK pathway, gene pairs found in both sets of analyses fit the information known about the pathway. Additionally, specific statistically supported hypotheses for potential new interactions were made. It is important to note that, due to the problem of multiple comparisons, statistical significance cannot be achieved in the above data set by considering all-by-all locus correlations in performing these cross-locus comparisons.

4.3. Exploration of CGH phenotype of ERBB2 in expression of RTK genes

The pathway representation can be used as a visual, as well as computational, tool. By coloring the nodes in the graph based on either data values (expression, copy

number, etc.) or derived statistical values (correlation, F-test, etc.), it is possible to see these values changing across many samples.

In a set of breast cancer cell lines (Neve, Chin et al. 2006), statistical analysis showed that the data set could be divided into two subclasses based on ERBB2 amplification status. For these cell lines, both aCGH and expression data were available. While the original statistical analysis was done on the aCGH data, the two classes can also be seen in the RTK pathway when it is colored based on expression (Figure 16).

In this analysis, the statistical relationship between expression levels and a phenotype (panel A, expression versus ERBB2 amplified phenotype) or the correlation between genes (panels B and C) can be illustrated. Here, the visualization lends support to the notion of a biological switch that delineates two states, which are marked by both ERBB2 amplification at the genomic level and a host of coordinated changes in gene expression.

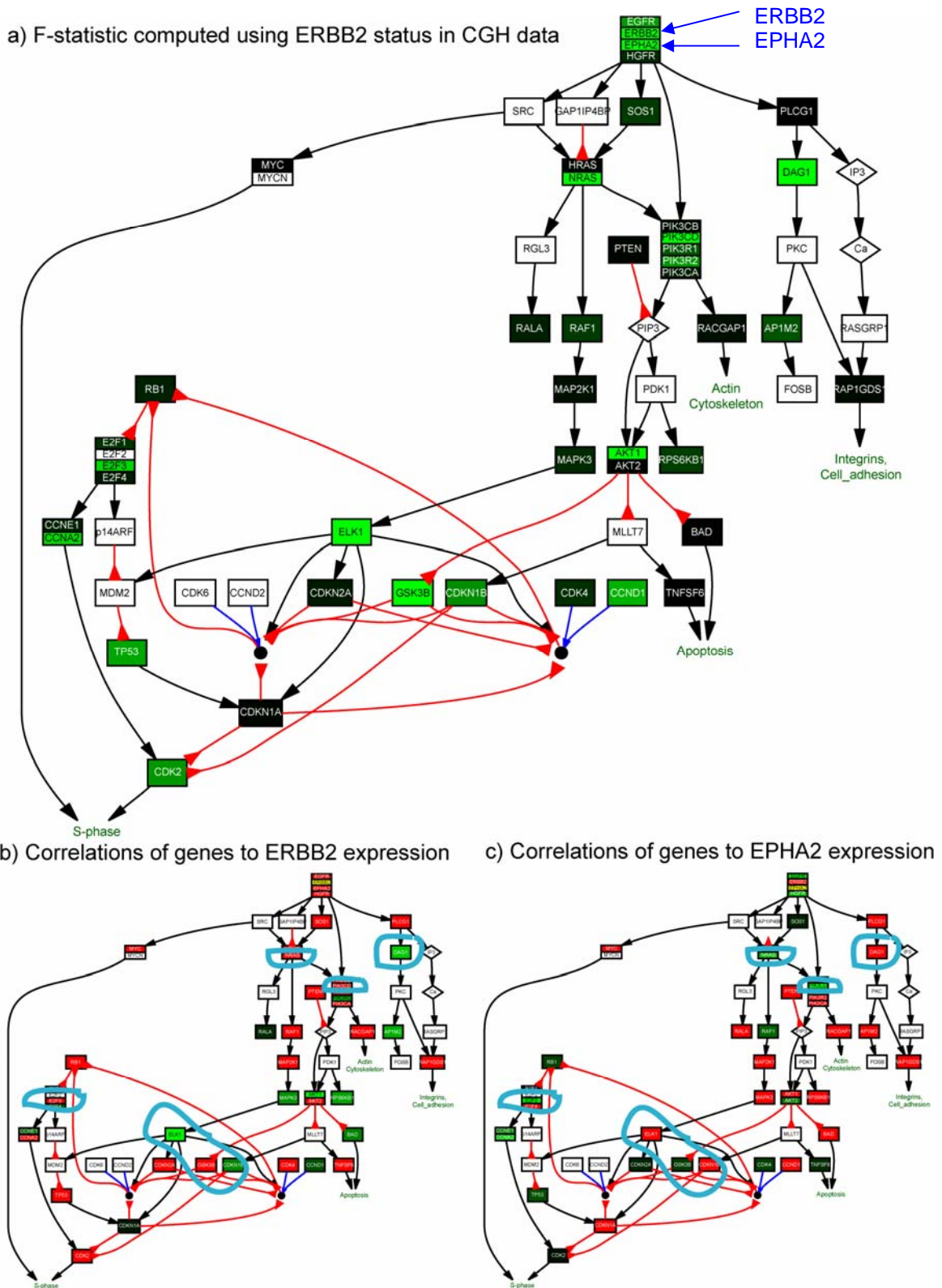


Figure 16. Visually differentiating between two classes of tumors.

In all images, white squares indicate missing values, from those genes either not being in the dataset or not passing the missing value threshold. Two classes of breast cell lines were previously identified using CGH data: ERBB2 amplifying and EPHA2 amplifying. a) The RTK pathway colored according to a computed F-statistic, which was based on these two classes. Brighter color indicates a higher discriminative value of the

gene. b) The RTK pathway colored according to each gene's correlation to ERBB2 using expression data. c) The RTK pathway colored according to each gene's correlation to EPHA2 using expression data. Images b and c are colored red/green with positive correlations in green and negative correlations in red. Uncorrelated genes are shown in black. All colors in the image are scaled to the maximum values of the statistic, i.e. the brightest colors correspond to the maximum actual value, not 1 or -1. The genes that showed a high discrimination ability between the two classes are outlined in blue. By comparing the correlation images to the f-statistic image, we can see that those genes with high f-statistics are highly correlated to either ERBB2 or EPHA2, and that these two genes are negatively correlated with each other.

4.4. Exploration of pancreatic cancer cell line data in the context of the RAS subpathway

A similar analysis to that in the previous example was also performed on expression data gathered from several pancreatic cancer cell lines (Gysin, Rickert et al. 2005). This analysis focuses on the RAS subnetwork (Figure 14) since the most frequently mutated genes in pancreatic cancer are RAS genes (70-90%), mainly *KRAS2*.

The data set consisted of 22 pancreatic cancer cell lines, both treated and untreated with CI-1040, a MEK1/2 specific inhibitor, as well as 3 cultures of normal pancreatic ductal epithelial cells (PDEC) run on the Affymetrix U133 2.0 array. As an initial overview of the dataset, the data for each sample was plotted onto the pathway representation, showing that the cell lines did not have equivalent reactions to application of the inhibitor. Some typical samples can be seen in Figure 17. Next, the Student's t-test was used to determine if any genes were able to discriminate between the treated and untreated samples (Figure 18). Those genes that were obviously differentially expressed in the sample overview (*MAP3K8* and *FOS*) were the only two that showed a statistically significant ability to discriminate between the treated and untreated samples.

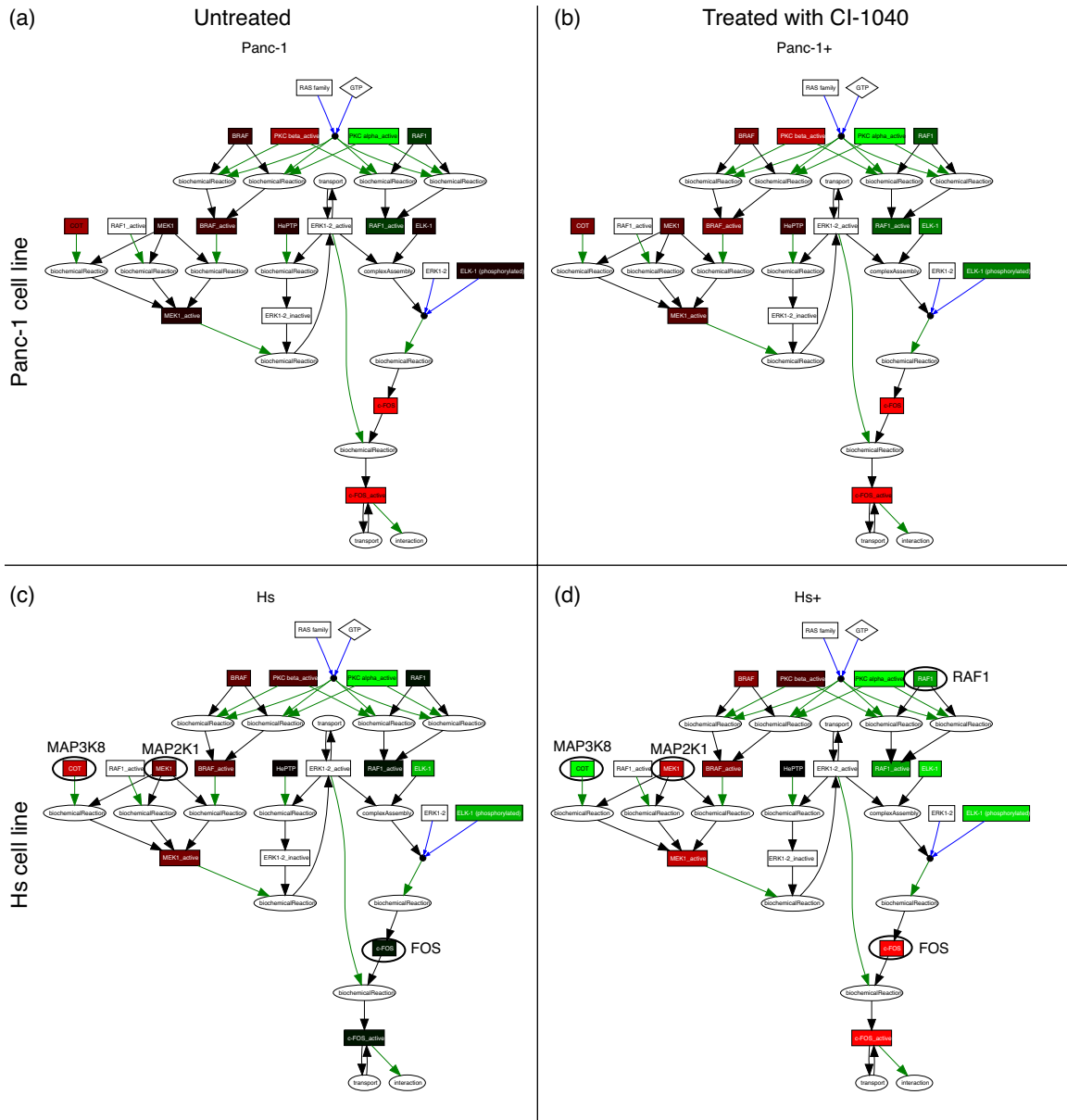


Figure 17. Gene expression differences in cell lines treated and untreated with a MAP2K1 inhibitor. Gene expression data for two typical cell lines, both treated and untreated with CI-1040, coloring the RAS subnetwork. Expression ratios were computed for each gene by comparing to the average expression value for that gene. Red indicates down-regulation compared to the average and green indicates up-regulation compared to average. (a) and (b) show a cell line (Panc-1) which did not show noticeable differences in expression when treated with the inhibitor. (c) and (d) show the Hs cell line, which shows a large difference in expression, especially of MAP3K8 (up-regulation) and FOS (down-regulation). There was also a slight down-regulation of MAP2K1 and a slight up-regulation of RAF1.

T-test between samples with and without CI1040

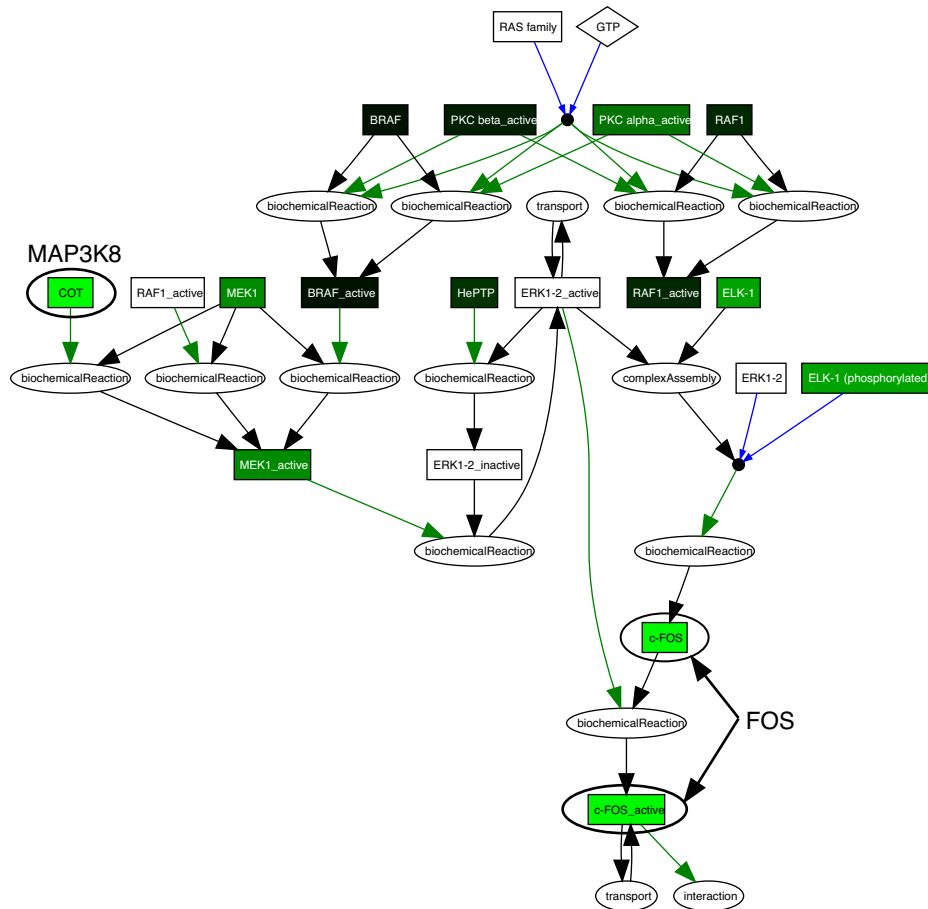


Figure 18. Discriminating between treated and untreated cell lines.

T-test p-values for each gene are indicated in shades of green, with brighter colors indicating more significant p-values. Both FOS and MAP3K8 (both colored in the brightest green) have p-values ≤ 0.05 .

Interactions in pathways frequently rely on post-translational modifications, such as phosphorylation in the case of MAP2K1 in the RAS subnetwork. Even though the inhibitor specifically targets MAP2K1, it is not unusual that this gene is not the most differentially expressed between treated and untreated groups. Instead, treatment with the inhibitor tends to up-regulate MAP3K8, which phosphorylates MAP2K1. Additionally, one of the downstream effects of MAP2K1 is activation of FOS transcription. In the treated cell lines, FOS is frequently down-regulated. Of note in this analysis is the observation that not all samples behaved similarly in response to the same treatment. The sample subselection method described in Chapter 3 addresses this exact issue and will be

discussed at greater length in the following chapter. This data set will also be revisited in Chapter 6 and Chapter 8.

4.5. Conclusion

Using QPACA to visually map both data and statistical values onto a visual representation of a pathway allowed for insights into several different types of data sets (both aCGH-based as well as expression-based studies). The context of the pathway was useful in both reducing the dimensionality of the data for improved statistical testing as well as for explanation of potential causality. The pathway visualization also reflects QPACA's ability to parse both locally defined pathways, as well as BioPAX formatted pathways downloaded from external databases.

Unlike other pathway visualization tools, QPACA also includes tools for displaying derived statistics (Sections 4.3 and 4.4) as well as tools for easily querying pathways for genes of potential interest (Section 4.2). The statistical tools allow biologists to make a quick qualitative assessment of the data in the context of the pathway, while the querying tools enable biologists to easily ask questions based on pathway structure.

The following chapters will deal with quantitative pathway analyses addressing the issues of both pathway recognition and pathway prediction.

Chapter 5

Computational Recognition of Pathways from Microarray Expression Data

5.1. Introduction

The previous chapter explored the use of QPACA as a qualitative analytical tool. This chapter introduces the use of QPACA as a quantitative analytical tool, focusing on addressing the issue of pathway recognition, which is the ability to discriminate between real pathways and non-pathways. In a biological context, this type of experiment can aid in answering the question of whether a set of genes identified through some procedure (such as differential expression between phenotypes) is likely to be part of a pathway or coordinated process, rather than simply a collection of genes that are parts of many tangentially related processes. Computationally, this type of experiment can be used as a feasibility proof for pathway augmentation. If a set of genes can confidently be identified as part of a pathway given an experimental data set, then it may be possible to make predictions of new pathway members given this data set. The following work was carried

out in this second context, as a stepping-stone to pathway member prediction, which will be further discussed in Chapter 6. Given that the results were positive, inquiries motivated by the former context are also sensible in applying QPACA.

Chapter 1 and Chapter 2 discussed the problems and promises of rapidly accumulating gene expression data and the current methods for investigating pathways using that data. QPACA's analysis method directly addresses the issues inherent in analysis of human systems without discretizing data or making limiting assumptions about the structure of pathways. Given a set of genes hypothesized to be part of a pathway or coordinated process, QPACA is able to reliably distinguish true pathways from non-pathways using microarray expression data. As discussed in Chapter 1 (Section 1.5.2), only some of the experiments within a large data set are necessarily relevant to a specific biochemical pathway. By leveraging the optimization method described in Chapter 3 (Section 3.3), QPACA identifies the particular samples in an experimental data set that are particularly relevant to a specified set of genes and in so doing also assigns a coordination score to this set of genes (see Figure 8). This coordination score can then be used to determine whether a gene set represents a pathway or coordinated process.

This chapter presents data on all human and yeast pathways found in the KEGG pathway database. In 117 out of 191 cases (61%), QPACA was able to correctly identify these positive cases as *bona fide* pathways with p-values measured using rigorous permutation analysis. Success in recognizing pathways was dependent on pathway size, with the largest quartile of pathways yielding 83% success.

This work appeared previously in Novak and Jain 2006.

5.2. Materials and Methods

5.2.1. *Data and data processing*

Both human and yeast expression data sets were considered in developing and assessing QPACA. The Hughes compendium of yeast data (Hughes, Marton et al. 2000)(287 deletion mutants and 13 environmental conditions) was used primarily to develop and test the algorithm, while the NCI set of 60 human cancer derived cell lines (Staunton, Slonim et al. 2001) was used to validate the results in human data. The data sets contained expression levels for 6356 and 7071 genes, respectively. The data were processed to remove duplicates, genes with excessive missing values, and genes with insufficient annotations. Because the algorithm computes correlations across at most 20 samples, a fairly stringent measure of missing values was used. Each gene had to have signal for at least 80% of the total samples. In the NCI60 data set, this threshold eliminated roughly 3600 genes. The other genes were eliminated due to insufficient annotation (~700 genes) and consolidation of duplicates (~800 genes). The yeast data set had fewer issues with missing annotations and duplicated genes, as well as better general signal strength. After processing, 6163 genes remained in the yeast data set, while 1954 genes remained in the human data set. Raw expression values were converted to log₂ ratios by taking the logarithm of each gene's value divided by the mean for that gene across all samples. Additionally, data were normalized by median centering within each sample (this is an automatic feature of QPACA's data processing, as it can influence results markedly if omitted). Both data sets were analyzed in the context of all pathways in the KEGG database with at least 5 genes in the corresponding data set. In the human data set, the receptor tyrosine kinase signaling (RTK) pathway, as defined by local

experts, was also analyzed. Figure 13 delineates the composition of this pathway. This resulted in a total of 83 yeast pathways and 108 human pathways. The pathways varied substantially in size, with the bottom half containing fewer than 20 genes, and the top quartile containing 30 genes or more.

5.2.2. *Permutation testing and p-value calculation*

The algorithm's ability to recognize pathway membership and to select samples relevant to a gene set was tested. Scores for each known pathway gene set (coupled with its experimental expression data) were compared to three distributions: 1) scores resulting from randomly chosen gene sets of the same size as the pathway set in question; and 2) scores resulting from randomizing the gene expression data itself; and 3) scores resulting from randomly chosen gene sets of the same size as the pathway set in question, restricted to genes represented within KEGG (see Figure 19 for a pseudo-code description of the methods). In each permutation test, the randomized data comes from the same microarray experiment as that used for the pathway case being tested. The first permutation method estimates the likelihood that any equal-sized gene set will find a sample subset which scores equal to or better than the test. If the probability is low, confidence of the metric's rationality increases. The second randomization method controls for the effects of the distribution of data values within a particular set of genes, which may yield unusual distributions of correlation values under certain degenerate conditions. Given a particular set of genes from a particular data set, data were randomized for each gene across all samples. The third method is a further control to test for the possibility that KEGG genes as a set may be biased.

Additionally, the optimized score($G_{\text{path}}, S_{\text{path}}, E$) was also compared to the unoptimized score(G_{path}, S, E) to determine if the optimization procedure actually increased our ability to recognize pathways.

method 1: randomize pathway gene set, G_{path} , without changing expression matrix, E

```

for i in (1:maxPermutations)
   $G_{\text{random}}$  := randomly choose gene set from  $G$  of same size as  $G_{\text{path}}$ 
   $S_{\text{final}}$  := optimize( $G_{\text{random}}, E$ )
  ScoreArray[i] := score( $G_{\text{random}}, S_{\text{final}}, E$ )

```

method 2: randomize expression matrix, E , without changing G_{path}

```

for i in (1:maxPermutations)
   $E'$  := randomized matrix  $E$ 
   $S_{\text{final}}$  := optimize( $G_{\text{path}}, E'$ )
  ScoreArray[i] := score( $G_{\text{path}}, S_{\text{final}}, E'$ )

```

method 3: same as method 1, but restrict G_{path} to genes found in KEGG pathways

Figure 19. Pseudo-code for permutation methods.

In each case, the p-value for G_{path} was computed by comparing the score($G_{\text{path}}, S_{\text{path}}, E$) to the distribution of the scores in ScoreArray where:

$$\text{p-value} := \text{count}(\text{ScoreArray}[i] > \text{score}(G_{\text{path}}, S_{\text{path}}, E)) / \text{maxPermutations}$$

In the experiments described, maxPermutations was set to 500.

5.3. Results

Representative results for gene set recognition on a diverse and non-overlapping set of yeast and human experimental data sets can be seen in Table 2 and Table 3.

Table 2. Results table for a representative set of pathways.

Organism	PathwayID	Pathway	Number of genes	Final score	p-value w/ subselection	p-value w/o subselection
a) Yeast	sce00052	Galactose metabolism	30 / 30	0.826	<<0.01	<<0.01
	sce00230	Purine metabolism	95 / 95	0.231	<0.01	0.03
	sce04110	Cell cycle	100 / 100	0.227	0.01	<0.01
	sce00100	Sterols biosynthesis	14 / 14	0.682	0.01	<<0.01
	sce04020	Second messenger signaling	19 / 19	0.497	0.04	0.26
	sce04010	MAPK signaling	55 / 55	0.221	0.11	0.09
	sce04070	Phosphatidylinositol signaling	12 / 12	0.376	0.60	0.59
	sce00562	Inositol phosphate metabolism	32 / 32	0.12	0.96	0.55
b) Human	hsa00010	Glycolysis / Gluconeogenesis	24 / 60	0.253	<<0.01	<<0.01
	hsa00052	Galactose metabolism	13 / 28	0.377	<0.01	0.01
	RTK	Receptor tyrosine kinase signaling	24 / 57	0.198	0.01	<<0.01
	hsa04510	Integrin-mediated cell adhesion	25 / 82	0.161	0.01	<<0.01
	hsa04010	MAPK signaling	41 / 287	0.1	0.01	0.09
	hsa04110	Cell cycle	30 / 120	0.131	0.02	0.05
	hsa04350	TGF-beta signaling	11 / 72	0.351	0.03	0.11r
	hsa04620	Toll-like receptor signaling	12 / 92	0.299	0.04	0.03
	hsa04210	Apoptosis	19 / 96	0.187	0.07	0.34
	hsa04070	Phosphatidylinositol signaling	13 / 70	0.217	0.29	0.47

Each pathway is listed with its KEGG id, if available. The number of genes indicates those genes that both belonged to the pathway and were represented in the expression dataset over the total number of genes in that pathway. The final score is the median of all gene-gene correlations after optimization. Two empirically derived p-values are listed: 1) the probability that the subselection algorithm gave a better score than random and 2) the probability that a better score than random could be found without subselection. The p-values reported represent the most pessimistic p-value derived from each of the three random permutations, which can be seen in Table 3.

Table 3. Breakdown of individual p-values with sample subselection for each pathway.

PathwayID	Randomized data p-value	Random gene set p-value	Random KEGG genes p-value
sce00052	<<0.01	<<0.01	<<0.01
sce00230	<<0.01	<<0.01	<0.01
sce04110	<<0.01	<<0.01	0.01
sce00100	<<0.01	0.01	<<0.01
sce04020	<<0.01	0.02	0.04
sce04010	<<0.01	<0.01	0.11
sce04070	0.03	0.60	0.02
sce00562	0.35	0.80	0.96
hsa00010	<<0.01	<<0.01	<<0.01
hsa00052	<<0.01	<0.01	<<0.01
RTK	<<0.01	<0.01	0.01
hsa04510	<0.01	0.01	<<0.01
hsa04010	<0.01	0.01	<0.01
hsa04110	<<0.01	0.02	0.01
hsa04350	<0.01	0.03	<<0.01
hsa04620	0.01	0.04	0.01
hsa04210	0.02	0.07	0.02
hsa04070	0.27	0.29	<<0.01

The individual p-values are: randomized data (scrambling the data for each gene in the pathway gene set), random gene set (gene sets of the same size as the pathway gene set randomly selected from the set of genes in the entire data set that are not part of the pathway gene set), and random KEGG gene set (gene sets of the same size as the pathway gene set randomly selected from the subset of genes in the entire data set that belong to other pathways in KEGG).

Table 2 reports the most pessimistic p-value computed from the three methods described above, while Table 3 provides the breakdown of all three individual p-values. A p-value of X can be interpreted as the probability that a random set of genes will yield a p-value of less than or equal to X . For this limited set of pathways, those for which subselection did not improve the significance of the score included the metabolic pathways (e.g. galactose metabolism) which are likely to show coordinated expression under many different conditions. In these pathways, the score was highly significant both with and without subselection. There were a few pathways, however, for which the score was not significant in either condition. In these cases (inositol phosphate metabolism and phosphatidylinositol signaling), it is possible that the perturbations in these pathways either did not show up in these particular data sets or that the effects were outside the scope of expression data, through post-translational modifications, for example. In none of these cases did the subselection process cause a significant result to become insignificant. Note, however, that significant variability in the p-values obtained using the three different methods was occasionally observed. These differences are expected, to the extent that each of the different permutation methods will yield different distributions of gene expression values. In the random data case, the distribution for the control is exactly the same as in the non-control. In the other two types of permutations, this is not so, and consequently, there were differences depending on the method used.

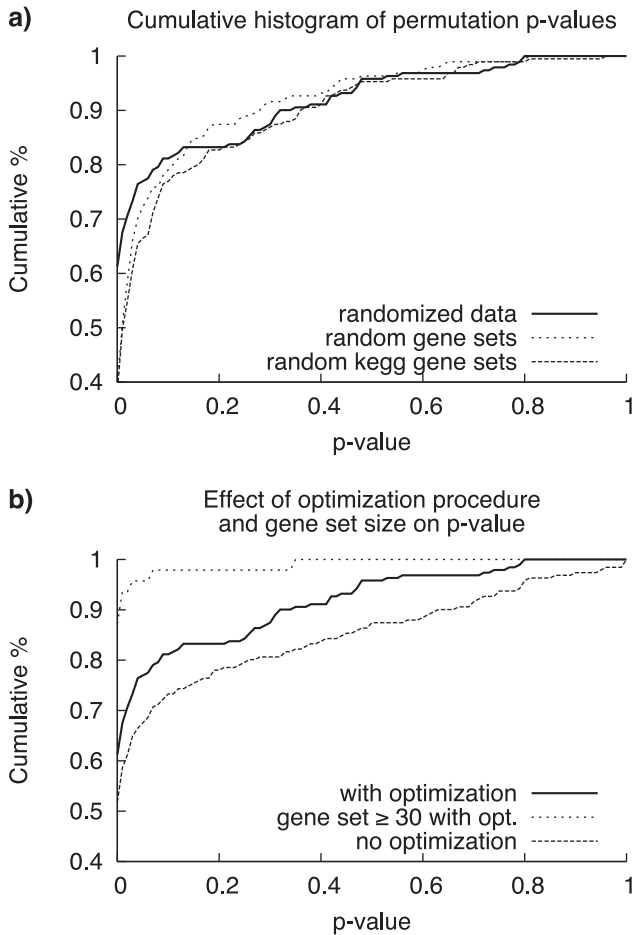


Figure 20. Comprehensive pathway recognition results from all 191 human and yeast pathways. (a) Cumulative histograms of permutation-based p-values for all analyzed pathways. Most pathways (117/191 or 61%) had significant scores ($p \leq 0.05$) under the most stringent of these methods, indicating that this method is reproducible over a large and varied set of pathways. (b) Cumulative histograms of the p-values with optimization for all pathways (red), all pathways without optimization (blue), and for those pathways with ≥ 30 genes using optimization (green). For each case, the randomization permutation method was used. The optimization method clearly improves the p-value distribution over no optimization ($p \ll 0.001$ by Mann-Whitney), and restriction to larger gene set sizes also improves the p-value distribution ($p \ll 0.001$ by Mann-Whitney).

Figure 20 illustrates the comprehensive results obtained from all 191 human and yeast pathways. Panel A shows the cumulative histograms for the three different permutation methods. While there are some differences in the distributions, particularly at very low p-values, the three methods do not produce substantially different interpretations of the data. Using the most pessimistic values (as in Table 2), in 61% of all cases, we observed significant scores under optimization ($p \leq 0.05$). For the reason of

distributional equivalence, we focused on the randomized data method. Panel B shows the effect of both the optimization procedure and larger gene set sizes for the data randomization permutation method. The optimization procedure clearly shifts the p-value distribution to the left, and the subset of pathways with larger gene set sizes also shifts the distribution to the left (both observations are highly statistically significant). Using the data randomization permutation method, for pathways with at least 30 genes, we observe 96% recognition at a false positive rate of 0.05. A more conservative test, using the most pessimistic of all permutation methods, yields a recognition rate of 83%.

5.4. Discussion and conclusion

The results detailed in this chapter address one aspect of computational structure elucidation using gene expression data. Given a gene set postulated to be part of a pathway or coordinated process, QPACA has demonstrated the ability to discriminate between real pathways and non-pathways. This problem is termed pathway recognition. Specifically, in 61% of cases known pathways (83% for larger pathways), QPACA was able to answer affirmatively while rejecting random pathways over 95% of the time.

By employing an automated method for identifying relevant sample subsets, QPACA has been able to analyze complex pathways relevant to major disease processes in humans. The next chapter covers the use of QPACA to address the other aspect of computational structure elucidation: pathway augmentation, or the ability to significantly enrich for pathway genes over non-pathway genes in making predictions about putative pathway members.

Chapter 6

Computational Prediction of New Pathway Members from Microarray Expression Data

6.1. Introduction

The previous chapter introduced the use of QPACA as a quantitative analytical tool in pathway recognition. Once a set of genes can reliably be distinguished as a true pathway or coordinated process, the next logical question is whether that set of genes can be augmented with new members.

This chapter will describe an approach to the pathway membership augmentation problem using the same optimization and scoring technique previously described for pathway recognition. Once a gene set has been recognized as a pathway or coordinated process, this method ranks non-pathway genes using the same scoring metric that was used for pathway recognition. Genes that increase the coordination score when added to the pathway gene set can reasonably be supposed to be related to the existing known pathway genes.

This technique was tested with a series of cross-validation experiments in which QPACA was able to yield enrichments for predicted pathway genes over random genes at rates of 2-fold or better the majority of the time, with rates of 10-fold or better 10–20% of the time.

6.1.1. *Cross-validation in biological pathway augmentation*

Cross-validation is a common statistical technique that involves partitioning a data set into training and validation sets. In the holdout method, some observations, usually less than 1/3, are randomly chosen to form the validation set while the remaining observations form the training data. In the case of the QPACA augmentation algorithm, a randomly chosen 10% of the known pathway genes were held out during the initial recognition scoring/optimization of the pathway gene set. The augmentation algorithm was then used to rank both the held-out and non-pathway genes. If the algorithm successfully identifies potential pathway members, then the held out pathway genes should rank highly compared to the non-pathway genes.

The general fuzziness and interconnectedness of existing pathway definitions makes forming a true validation set of known non-pathway genes real data difficult. Certain broad generalizations can, however, be made. It is possible to subdivide the negative (i.e., non-pathway genes) into several general groups: (a) genes not known to belong to any pathway, (b) genes in other known KEGG pathways, and (c) genes in closely related pathways. It would be expected that enrichment for the held out known pathway genes would be greatest against the background of set (a) and least against that of set (c).

Construction of this third background of interrelated pathways was accomplished by subsetting large pathways. To some degree, all processes within an organism are interrelated. As a result, many of the most intensively studied pathways in existing pathway databases contain large numbers of genes (some number in the hundreds) and many different subprocesses. The RTK pathway in Figure 13, for example, contains several distinct processes. By choosing gene subsets of these large pathways, it was possible to examine the effect of genes in closely related pathways on pathway augmentation.

This chapter will first introduce the augmentation algorithm and scoring metric as well as the data sets used in the analysis. Then, it will discuss the results of the cross-validation in two separate sections: first focusing on the data sets used in Chapter 5 and the first two backgrounds described above and then focusing on an ovarian data set and a background of closely related subpathways.

Portions of this work on cross-validation in the human and yeast data sets presented in Chapter 5 appeared previously in Novak and Jain 2006.

6.2. Materials and Methods

6.2.1. *Augmentation algorithm and scoring metrics*

The method used for pathway augmentation is described in Figure 21. In essence, each gene is scored based on the recognition score for the pathway with that gene added. Recall that the original scoring metric for pathway recognition was the median of all gene-gene correlations within the gene set (Figure 8). Unfortunately, due to its reliance on medians, this method results in a fair number of identical scores. In order to reduce the

number of ties, another scoring metric was also tested: the average of the top 10% of all gene-gene correlations within the gene set.

```
procedure augmentPathway( $G_{\text{path}}, G_{\text{notPath}}, E$ )
   $s_{\text{path}} = \text{optimize}(G_{\text{path}}, E)$ 
   $\text{pathScore} = \text{score}(G_{\text{path}}, s_{\text{path}}, E)$ 
  foreach gene  $g$  in  $G_{\text{notPath}}$ 
     $G_{\text{test}} = G_{\text{path}} \cup g$ 
     $g_{\text{score}} = \text{score}(G_{\text{test}}, s_{\text{path}}, E)$ 
    if ( $g_{\text{score}} > \text{pathScore}$ )
       $\text{results}[g] = g_{\text{score}}$ 
end
```

Figure 21. Pseudo-code for augmentation algorithm.

Only those genes that improve the score for the pathway gene set are likely candidates for pathway membership.

6.2.2. *Data and data processing*

Three data sets were used to explore the utility of this augmentation algorithm. The initial set of experiments was conducted using the two data sets described in Chapter 5 (NCI60 cell line data and Hughes yeast compendium data). The 101 pathways for which all three permutation-based recognition p-values were less than 0.1 were used, eliminating those pathways for which prediction would be unlikely to work. This filtering matches the way in which a researcher would proceed in practice, where predicted augmentations of a pathway in which the recognition process failed would not be believed.

Sets of closely related subpathways were constructed using a data set of serous ovarian tumors. This data set included CGH as well as expression data, allowing for the formation of logical pathway submodules. The data set consisted of 50 snap-frozen serous ovarian tumors from the UAB/Duke Ovarian Cancer SPORE, comprised of 23 patients that survived >60 months (long survivors) and 27 that survived <36 months (short survivors)(Berchuck, Iversen et al. 2005). Samples in the data set were median-

centered and had a log₂ ratio calculated for each value by taking the value divided by the average of all values for that gene and then taking the log base 2 of this ratio. To account for outliers that affect the calculation of Pearson's correlation, the data were assigned ranks prior to analysis. Specific pathways were chosen for the analysis from the KEGG database (Kanehisa and Goto 2000) that focused on EGFR, PI3K and TGF-beta, which are all implicated in ovarian cancer. These pathways were: MAPK signaling, calcium signaling, cytokine-cytokine receptor interaction, phosphatidylinositol signaling, cell cycle, TGF-beta signaling, focal adhesion, gap junction, and actin cytoskeleton regulation.

6.3. Cross-validation of pathway augmentation using human and yeast microarray data

For the 101 pathways from Chapter 5 where QPACA recognized the gene sets as pathways, cross-validation experiments were performed by holding out random subsets of known pathway genes and using QPACA's scoring methodology to rank-order the held-out pathway genes within a background of non-pathway genes (see Section 6.1.1 for additional details). Figure 22 summarizes the results using two background gene sets: one consisting of only of genes found in KEGG and the other of non-KEGG genes. Panel A shows smoothed histograms of scores for holdout RTK pathway genes and non-pathway genes (KEGG control gene set). The distributions are roughly normal, with the scores for the RTK pathway genes clearly shifted to the right. The RTK case was fairly typical in terms of enrichment of high scores within the pathway holdout set versus the non-pathway set. Panel B shows the corresponding representative receiver-operating characteristic (ROC) curve for the RTK pathway and for two other human pathways.

Panels C and D of Figure 22 summarize cross-validation results for the full set of 101 tested pathways with two different backgrounds. In Panel C, the cumulative distributions of ROC areas for the two control gene sets are shown. Both distributions are highly skewed to the right of 0.5 ($p < 10^{-6}$ by exact binomial), indicating that positive enrichment for pathway genes over non-pathway genes occurs in the vast majority of cases. The non-KEGG control gene population yields a slightly stronger enrichment. Panel D shows cumulative histograms of maximal enrichment ratios for both control gene populations. Given a specific portion of the top-ranked genes in a ranked list, the enrichment reflects the ratio of the number of true positive genes found divided by the number of such genes expected by chance. The maximal enrichment ratio is the highest such ratio over the full range of possible proportions. Enrichments of 2-fold or better occur for 60% and 70% of the pathways using the KEGG and non-KEGG control gene sets, respectively. Enrichments of 10-fold or better occur in 12% and 20%, respectively. Note that all of the results in Figure 22 resulted from running the subset optimization procedure described previously. On the same set of 101 pathways, without running the optimization procedure, the chief difference in the results is a marked increase in the number of pathways for which the resulting ROC areas were significantly less than 0.5 (7% of non-optimized cross-validation runs yielded ROC areas < 0.485 compared with 0% of the runs with optimization).

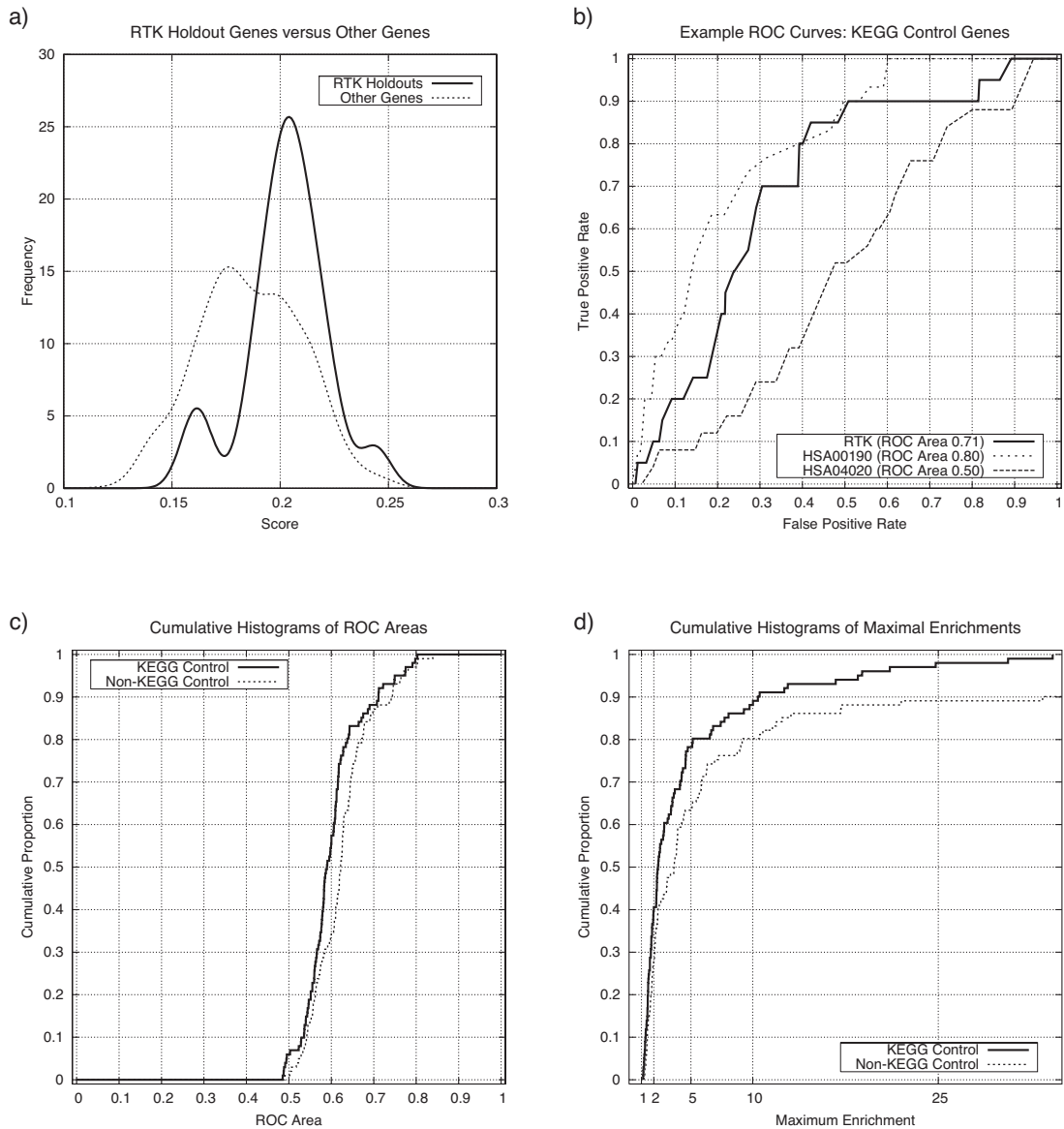


Figure 22. Comprehensive pathway augmentation results from 101 human and yeast pathways. (a) Smoothed histogram of scores for holdout RTK pathway genes (solid line) and non-pathway genes (dashed line). The non-RTK-pathway genes were taken from the full set of KEGG genes that were not found in the annotated RTK pathway. The distributions are roughly normal, with the scores for the RTK pathway genes shifted to the right. (b) Three representative receiver-operating characteristic curves: RTK pathway (solid line), two KEGG human pathways (dashed lines). The shift of the RTK holdout distribution to the right of the non-pathway gene distribution yields a favorable ROC curve, with deflection into the upper left corner, with an area of 0.71. HSA00190 (Oxidative phosphorylation) also shows a favorable enrichment of known pathway genes against a background of non-pathway genes. HSA04020 (calcium signaling pathway) shows an atypical case, with no enrichment of pathway genes over non-pathway genes. (c) Cumulative distributions of ROC areas for all 101 pathways tested with two different backgrounds. The solid line shows results using the KEGG gene set as the source of non-pathway genes for each pathway holdout experiment. The dashed line shows similar results, but makes use of genes not annotated within KEGG as the source of non-pathway genes. Both distributions are highly skewed to the right of 0.5, indicating that positive enrichment for pathway genes over non-pathway genes occurs in the vast majority of cases. The non-KEGG control gene population yields a slightly stronger enrichment. (d) Cumulative histograms of maximal enrichment ratios for both control gene populations. Enrichments of 2-fold or better

occur for 60% and 70% of the pathways using the KEGG and non-KEGG control gene sets, respectively. Enrichments of 10-fold or better occur in 12% and 20%, respectively.

Recall that the RTK pathway (Figure 13) was a hand-curated representation of genes downstream of receptor-tyrosine-kinase signaling, particularly emphasizing pathways that affected cell-cycle regulation, proliferation, and apoptosis (each particularly important in cancer). Also, note the nominally better enrichments observed in using the non-KEGG genes as a control set as compared with the KEGG gene set. It is possible that the high-scoring control genes from the KEGG population of nominally non-RTK genes might, in fact, include genes that properly belonged to the RTK pathway, based on existing literature evidence. The top 30 highest-scoring genes (the right-hand tail of the “Other Genes” distribution from Figure 22a) were examined. Figure 23 depicts the documented relationships of 12 of these genes to the RTK pathway as originally curated. 15/30 of the highest-scoring KEGG genes (all nominal false positives in our cross-validation experiment) properly belong to the RTK or very closely related pathways.

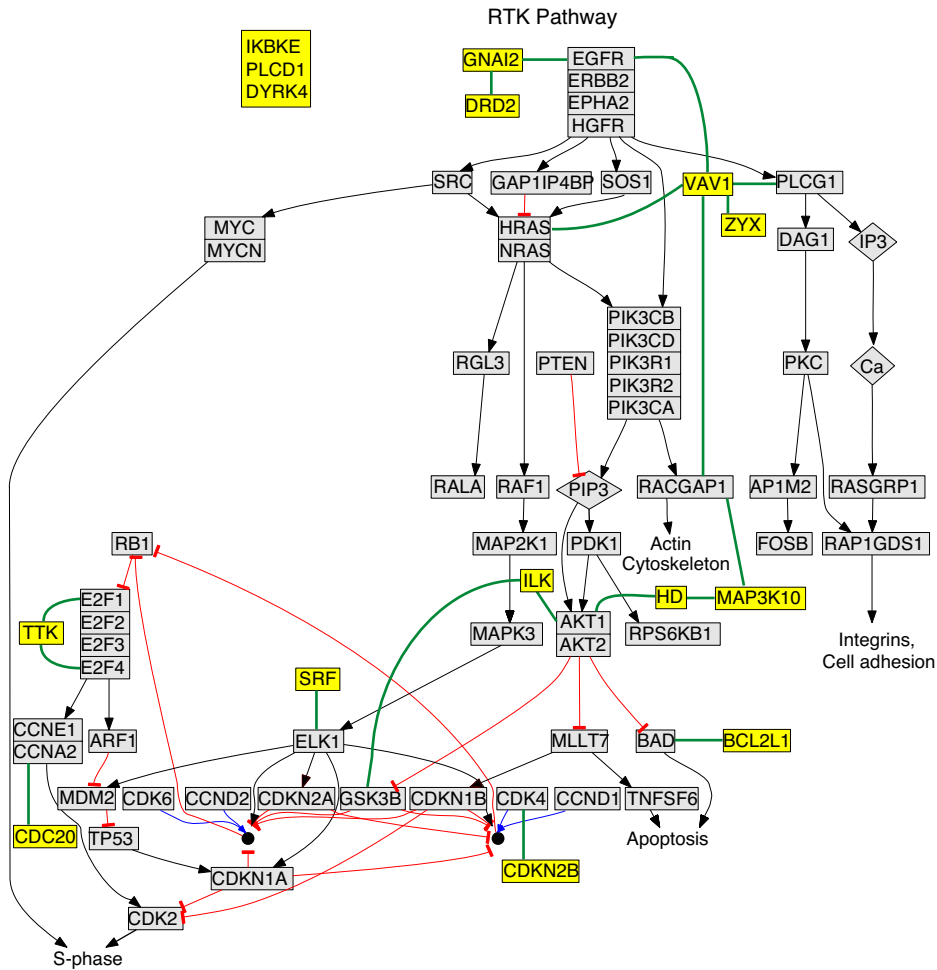


Figure 23. Pathway augmentation for the RTK pathway.

This image shows the original annotated RTK pathway as in Figure 13 with the 15/30 top scoring genes that have documented links to the pathway. Interactions between highly scoring genes (yellow) with annotated RTK pathway genes (gray) are highlighted in green (Gulbins, Schlottmann et al. 1995; Hall, Bates et al. 1995; Yang, Zha et al. 1995; Hobert, Schilling et al. 1996; Watson, Robinson et al. 1997; Bertagnolo, Marchisio et al. 1998; Delcommenne, Tan et al. 1998; Nagata, Puls et al. 1998; Das, Shu et al. 2000; Liu, Dorow et al. 2000; Moores, Selfors et al. 2000; Ohtoshi, Maeda et al. 2000; Zhang, Ho et al. 2001; Humbert, Bryson et al. 2002; Weinmann, Yan et al. 2002). Three additional genes (IKBKE, PLCD1, and DYRK4) are shown unconnected to the existing diagram. These genes were members of closely related pathways (MAPK signaling, Calcium signaling and Phosphatidylinositol signaling). DYRK4 is also a tyrosine kinase (Becker, Weber et al. 1998; Zhang, Li et al. 2005).

6.4. Cross-validation of pathway augmentation using closely related subpathways

As discussed in section 6.1.1, all processes within an organism are interrelated to some degree. As a result, what KEGG curators have called a "pathway", especially in the case of the larger pathways containing hundreds of genes, consists of modules that may

share a connection, but which may not behave as a coordinated process. In fact, when looking at heatmaps of expression data (such as Figure 24), it is possible to see distinct clusters of coordinated genes within the many genes in a large pathway.

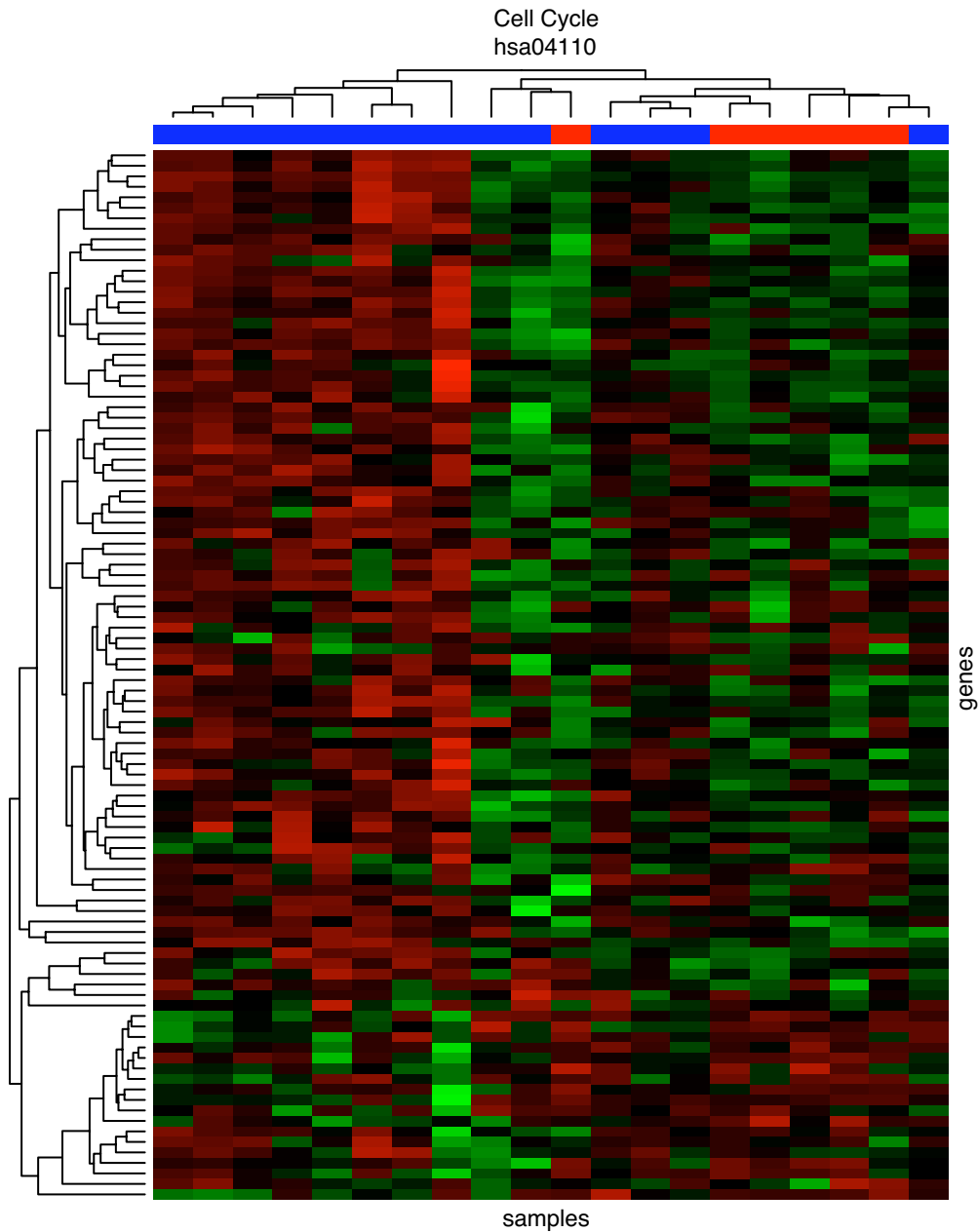


Figure 24. Expression heatmap of Cell Cycle pathway. Image of ovarian data for the KEGG Cell Cycle pathway (hsa04110) with over-expression colored in green and under-expression colored in red. The image shows all of the genes in the pathway but only the 20 samples that most contribute to the coordination score as chosen by the QPACA optimization algorithm during pathway recognition. The red and blue colored bar at the top of the heatmap indicates survival status: short survivors in blue and long survivors in red. The genes in this pathway can be seen to form at least two fairly distinct coordination modules, divided into a large module which is largely down-regulated

in the left half of the samples (all short survivors) and up-regulated on the right (mainly long survivors). There is a smaller cluster of genes at the bottom that shows the opposite behavior.

In light of these observations, it makes sense to consider smaller subsets of the pathway genes. These gene subsets also enable the formation of subpathways that are known to be closely related and can be used to further examine the efficacy of pathway membership augmentation using the QPACA algorithm.

This analysis will focus on subpathways in the ovarian data set that were formed based on both the defined structure of the pathway as well demonstrated relevance to ovarian cancer based on aCGH data.

6.4.1. Subpathway creation

Subpathways were defined based on the super-pathway's structure and a set of "core" genes. Since the data set contained both aCGH and expression data, it was possible to rigorously select several biologically interesting genes that were both highly deregulated on the genomic level in ovarian cancer and had expression levels that strongly correlated with the copy number variation. The first of these criteria ensures that the chosen genes are relevant to the genomic disturbances specific to ovarian cancer. The second criteria guarantees that the expression levels for these genes are related to these genomic disturbances.

Specifically, genes with (1) a corresponding CGH clone that had an absolute difference value of ≥ 0.2 in at least 30 samples and (2) an absolute value of the Pearson's correlation of aCGH to expression data of at least 0.5 were chosen. These selection criteria resulted in a list of 613 Affymetrix identifiers, corresponding to 443 unique annotated genes. Each of the selected pathways contained between one and four of these genes (see Table 4 for a complete list of pathways and core genes). Neighborhoods of 25-

60 genes (depending on the density of the surrounding pathway interactions) were chosen around these genes and defined as "subpathways". A sample image of the cell cycle pathway, highlighting the subpathways, can be seen in Figure 25. Core genes are highlighted in red.

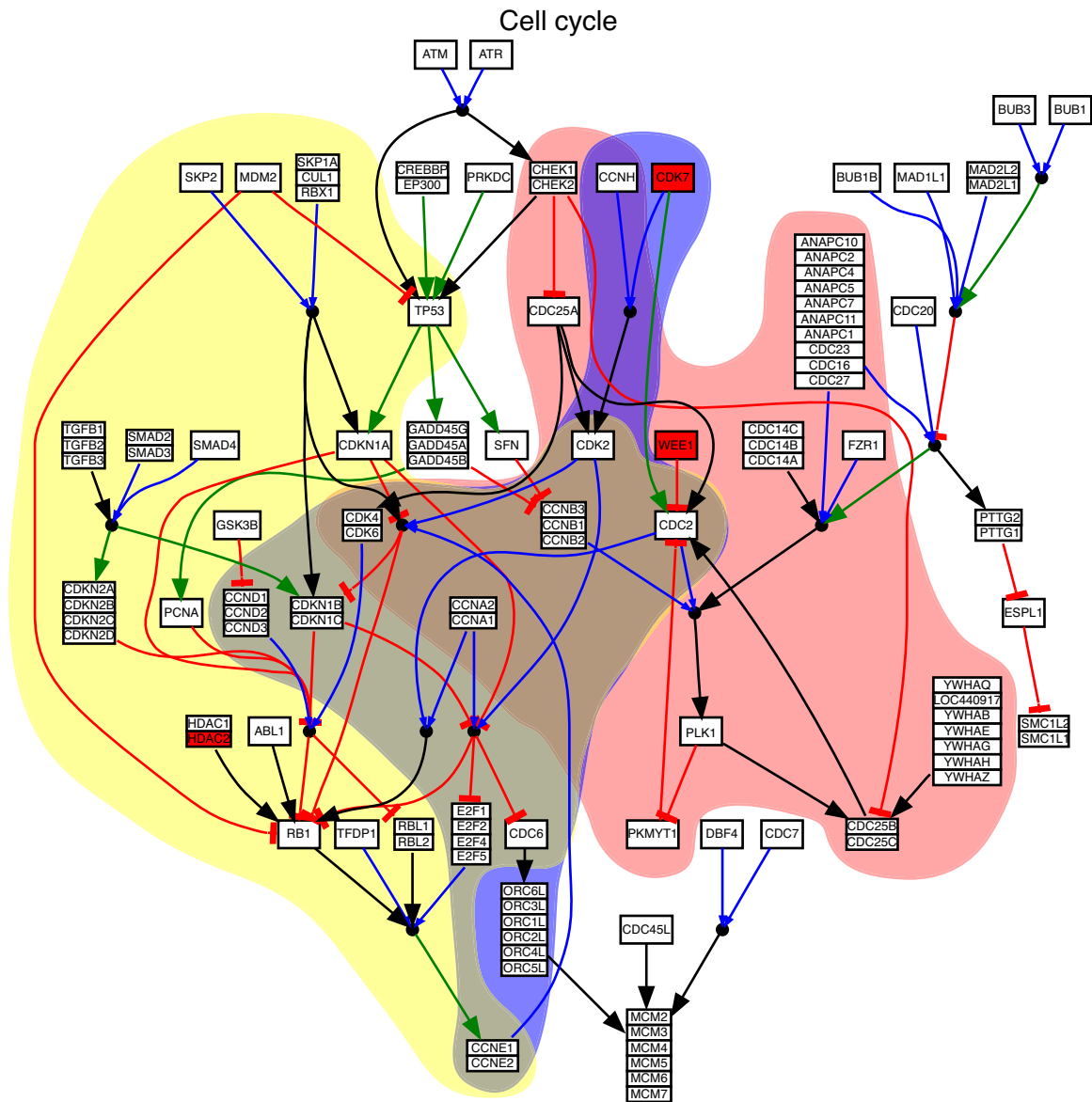


Figure 25. Subpathways in the Cell Cycle pathway (hsa04110).

The large shaded areas indicate subpathways centered on 3 core genes (HDAC2, WEE1, and CDK7), which are colored red. The core genes are both highly deregulated at the genomic level in ovarian cancer and have expression levels that correlate strongly with the copy number variation. The subpathway centering on WEE1 is colored in pink; the one centering on HDAC2 is in yellow, and the one centering on CDK7 is in blue. The entire pathway as well as all three subpathways were correctly recognized as being significantly coordinated using the ovarian cancer expression data set, making them good candidates for exploration of QPACA's predictive ability.

6.4.2. Results

Prior to examining pathway augmentation in these subpathways, it is always necessary to determine pathway recognition using the data set. In the subpathways based on core genes, 76% had significant recognition scores, which is consistent with the proportion observed in the Chapter 5. Of note is that some subpathways derived from the same super-pathways are unrelated to each other in the sense of co-regulation. This observation is not unexpected given both the distinct coordination modules and the existence of distinct subprocesses in large curated pathways. Just as some samples in a large data set do not contribute equally to strong coordination between the genes, some genes in a large pathway may not show variance on the expression level. Results of the analysis can be seen in Table 4.

Table 4. Results for pathway recognition in subpathways.

a) Super pathway name	b) Core gene name	c) Genes in subpath	d) Recog. p-value	e) Recog. score
hsa04010: MAPK signaling	KRAS	45/54	0	0.115
	PAK2	30/34	0	0.154
	MAP2K7	26/28	0.024	0.125
hsa04020: Calcium signaling	ATP2B4	50/56	0.306	0.044
	RYR1	39/44	0	0.205
hsa04060: Cytokine-cytokine receptor interaction	TNFRSF10B	7/8	0.054	0.455
hsa04070: Phosphatidylinositol signaling	BCR, DYRK4	23/27	0.032	0.136
	PIP5K1A, PIK4CB	27/29	0.02	0.119
hsa04110: Cell cycle	CDK7	32/33	0	0.343
	HDAC2	36/38	0	0.274
	WEE1	32/38	0	0.413
hsa04350: TGF-beta signaling	PPP2CB	36/41	0	0.174
hsa04510: Focal adhesion	KRAS	31/36	0	0.132
	PAK2	32/38	0	0.121
	PTK2	46/49	0.32	0.049
	CAPNS1	42/46	0.118	0.062
hsa04540: Gap junction	KRAS	34/38	0.002	0.126
	KRAS	37/39	0.01	0.088
hsa04810: Actin cytoskeleton regulation	PAK2	22/29	0.542	0.095
	PTK2	30/36	0	0.161
	PIP5K1A	25/31	0.002	0.151

Column (a) gives the pathway ID and name; column (b) specifies the subpathway core gene; column (c) indicates the number of genes that both belonged to the pathway and were represented in the expression data set over the total number of genes in the pathway; column (d) lists the p-value for pathway recognition and column (e) lists the recognition score.

Once a pathway has been determined to exhibit coordinated behavior based on a data set, non-pathway genes can then be assigned scores using the pathway augmentation algorithm (Figure 21). As in the previous cross-validation experiments in human and yeast data described above, holdout experiments were run on the subpathways in which a random 10% of genes were heldout during the optimization step. Then these heldout genes were scored along with the non-pathway genes. Enrichment was observed for known subpathway genes (Figure 26).

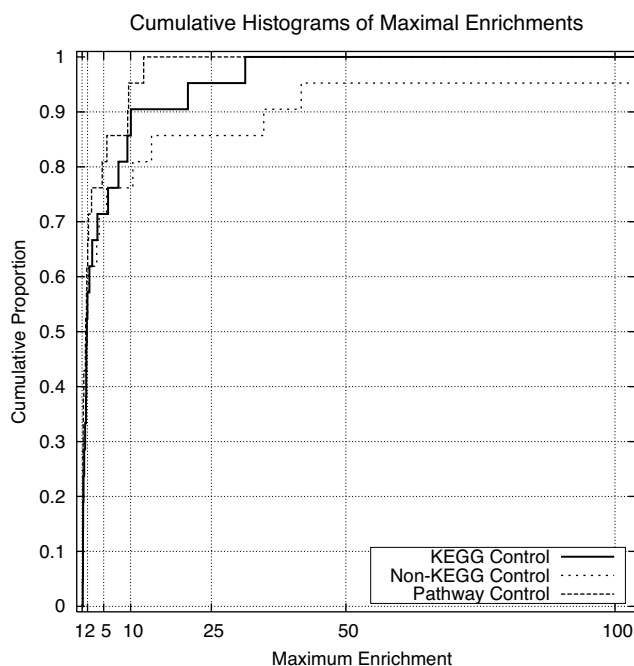


Figure 26. Comprehensive pathway augmentation results for all subpathways. Cumulative histograms of maximal enrichment ratios for all control gene populations: all genes found in the associated superpathway (dashed line), all genes found in other KEGG pathways (solid line), and all genes not found in KEGG (dotted line). Enrichments of 2-fold or better occur for about 40% of the pathways using any of the gene sets. Enrichments of 10-fold or better occur in 5% of the pathway gene set, 14% of the KEGG gene set and 24% of the non-KEGG gene set.

Of note is the effect of testing against a background of closely related subpathways (denoted as "Pathway Control"). As predicted, the enrichment for heldout genes is lowest against this background, indicating that the more closely related a gene is to the pathway, the more likely it is that it will have a high coordination score.

6.5. Conclusion

Once a set of genes is known to represent a pathway and is recognized given a particular dataset, the next obvious avenue of research is computational augmentation of this pathway. This chapter addressed this problem and demonstrated QPACA's ability to significantly enrich for pathway genes over non-pathway genes in making predictions of putative pathway members. QPACA was able to yield enrichments for predicted pathway genes over random genes at rates of 2-fold or better the majority of the time, with rates of 10-fold or better 10–20% of the time, depending on the control gene set used. It was compared to three different control gene sets (genes not known to belong to any pathway, genes belonging to other pathways, and genes belonging to closely related subpathways) in several different data sets. While enrichment rates on the level of 2-fold may or may not be of practical significance, rates of 10-fold or higher are more likely to help prioritize experimental exploration of candidate genes for elaborating pathways of particular biological interest. Of particular note is the observation that genes known to be closely related to a pathway are more likely to score highly under the pathway augmentation algorithm.

The next chapter will explore the synergy that arises by combining sequence-based analytical methods (specifically, motif finding in gene promoter sets) with the information derived from measurements of variation in gene expression.

Chapter 7 The Relevance of Sequence Patterns to QPACA

The previous two chapters addressed two steps in pathway structure elucidation using gene expression data. First, given a gene set postulated to be part of a pathway or coordinated process, QPACA demonstrated the ability to discriminate between real pathways and non-pathways. Second, given a gene set known to represent part of a pathway, QPACA was then able to significantly enrich for pathway genes over non-pathway genes in making predictions of putative pathway members. The first problem was termed pathway recognition and the second pathway augmentation. Specifically, for the pathway recognition problem, in 61% of cases known pathways (83% for larger pathways), QPACA was able to answer affirmatively while rejecting random pathways over 95% of the time. For the pathway augmentation problem, QPACA was able to yield enrichments for predicted pathway genes over random genes at rates of 2-fold or better the majority of the time, with rates of 10-fold or better 10–20% of the time depending on the background used. While enrichment rates on the level of 2-fold may or may not be of practical significance, it is likely that rates of 10-fold or better may help prioritize experimental exploration of candidate genes for elaborating pathways of particular biological interest. Additionally, Chapter 6 also introduced the idea that gene sets are

recognized as pathways not because all genes within them are cross-correlated, but rather because individual genes grouped together to form highly coordinated expression modules.

The surprising feature of the results from the previous two chapters is that the results for pathway recognition and augmentation were made possible based on the patterns of variation in gene expression alone. No information regarding protein levels, phosphorylation states, or pairwise physical interactions was required. The natural question is *why* it was possible to deduce aspects of pathway structure, as defined by human curators based on knowledge of the experimental biological literature, from expression patterns. The most appealing answer is that the transcriptional programs that govern expression behave in a manner that frequently yields experimentally detectable patterns of co-variation among gene sets within pathways. If this were so, it should be possible to identify motifs and/or specific transcription factors that are present, at least within subsets of pathway genes that themselves exhibit the coordinated behavior that QPACA detects in differentiating true pathways from non-pathways and in yielding better than random enrichments in producing ranked lists of potential new pathway members.

In this chapter, this avenue is explored by combining sequence-based analytical methods (e.g. motif finding in gene promoter sets) with the information derived from measurements of variation in gene expression or protein levels. The remainder of this chapter will describe some preliminary data that supports the idea that simultaneously considering transcription factor analysis and pathway gene expression coordination may enhance the ability to make novel pathway membership predictions.

7.1. Introduction

As discussed previously in Chapter 6, gene sets are recognized as pathways not because all genes in them are cross-correlated, but because some genes form highly coordinated expression modules. These gene modules are evident in Figure 24.

Since this phenomenon does not appear to be strongly related to the curated pathway structure, the obvious question is which biological process or processes direct the expression clustering. Expression data directly measures the relative transcription of genes into mRNA, and it is reasonable to assume that transcription factors, which are instrumental to this process, may play a role in the composition of the expression modules. Common, and perhaps overlapping, transcription factors driving different genes could account for this observation.

To examine this hypothesis, a method for deterministic motif identification was used to identify putative transcription factor binding sites in the expression gene clusters. Several motifs known to affect the genes in question were identified. Additionally, tightly clustered genes were better at finding consistent motifs than randomly selected gene sets.

7.2. Methods

7.2.1. *Data*

Two separate data sets were used in this analysis: the ovarian data set described in Chapter 6 and the pancreatic cancer cell line data set introduced in Chapter 4, section 4.4. Recall that the pancreatic data set consisted of cell lines that were both treated and untreated with CI-1040, a selective MAP2K1 and MAP2K2 inhibitor.

The ovarian data set analysis focused on the Cell Cycle pathway, which had also been examined using the subpathway analysis in Chapter 6. The ovarian data set allowed a comparison to be made between using pathway structure-based gene sets (subpathways, see Section 6.4.1) and expression-based gene clusters.

The pancreatic data set was analyzed in the context of several pathways, chosen specifically for their relevance both to MAPK signaling as well as two well-characterized transcription factors: E2F and NF-kappaB. These pathways included MAPK signaling (hsa04010), natural killer cell mediated cytotoxicity (hsa04650), focal adhesion (hsa04510), cell cycle (hsa04010), and cytokine-cytokine receptor interaction (hsa04060). Focusing on pathways that incorporate well-known transcription factors allowed direct comparison of the MaMF derived motifs to the known motifs.

7.2.2. Cluster formation

First, the QPACA pathway recognition algorithm was run to generate a set of samples particularly relevant to overall coordination within the entire pathway gene set. Then, gene sets representing distinct expression modules were created using K-means clustering on the data sets limited to the selected samples. In the Cell Cycle pathway using the ovarian data set, for example, six clusters were formed (Table 5).

The number of clusters and the cluster size were constrained by several practical considerations. For this analysis, MaMF (Hon and Jain 2006) was used for motif identification. MaMF was chosen for its performance in mammalian motif finding, and in particular because it was designed for and evaluated against human data. MaMF requires that the input gene sets have at least three genes in order to form a coherent result. In addition, due to computational limitations, clusters of more than 50 genes were split into

smaller clusters. Given a set of promoter sequences, MaMF produces a ranked list of putative motifs, each with a numerical score that combines sequence similarity among the subsequences comprising the motif with the uniqueness of the subsequences given the background distribution of sequences within the organism under study.

As a result, each pathway split into different numbers of clusters to stay within these parameters (see Table 5 for a breakdown of pathway and cluster composition).

Table 5. Pathway clusters.

Data set	Pathway	Genes in pathway	Number of clusters
Ovarian tumors	Cell cycle (hsa04110)	102	6
	Cell cycle (hsa04110)	102	5
Pancreatic cell lines	MAPK signaling (hsa04010)	236	13
	Natural killer cell mediated cytotoxicity (hsa04650)	61	6
	Focal adhesion (hsa04510)	154	11
	Cytokine-cytokine receptor interaction (hsa04060)	230	19

The larger the pathway, the more clusters needed to be created. The discrepancy between the number of clusters for the Cell Cycle pathway in the ovarian data set versus the pancreatic data set is attributable to the fact that dividing the genes into 6 clusters using the pancreatic data set resulted in a cluster that contained only one gene.

7.2.3. *MaMF setup and evaluation*

MaMF was used to find putative transcription factor binding sites that were common to each cluster of genes or subpathway. MaMF was used with the default set of parameters as described in (Hon and Jain 2006). Specifically, the following parameters were used: the width of the motif search was set to 11 bp; the nmer size used was 4 bp; and the number of seeds was set to 1000. Gene promoter sequences were gathered from the Database of Transcription Start Sites (DBTSS), Version 5.2.0 (Yamashita, Suzuki et al. 2006). The 14628 unique sequences consisted of -1000 and +200 bp around the transcription start site. This version was based on the UCSC hg17 genome sequence release and had already been repeat masked. A background distribution was constructed by MaMF based on the count with mutations method as used in (Hon and Jain 2006).

Putative binding sites for pathway based gene sets were evaluated in two ways: 1) qualitative comparison to known binding motifs found in the TransFac database of transcription factors (Matys, Kel-Margoulis et al. 2006) and 2) quantitative comparison to randomly selected gene sets. The random gene sets were selected to have the same number of genes as the pathway-based gene sets with no overlap between the two.

7.3. Results and discussion

7.3.1. Analysis of E2F binding sites in ovarian tumors

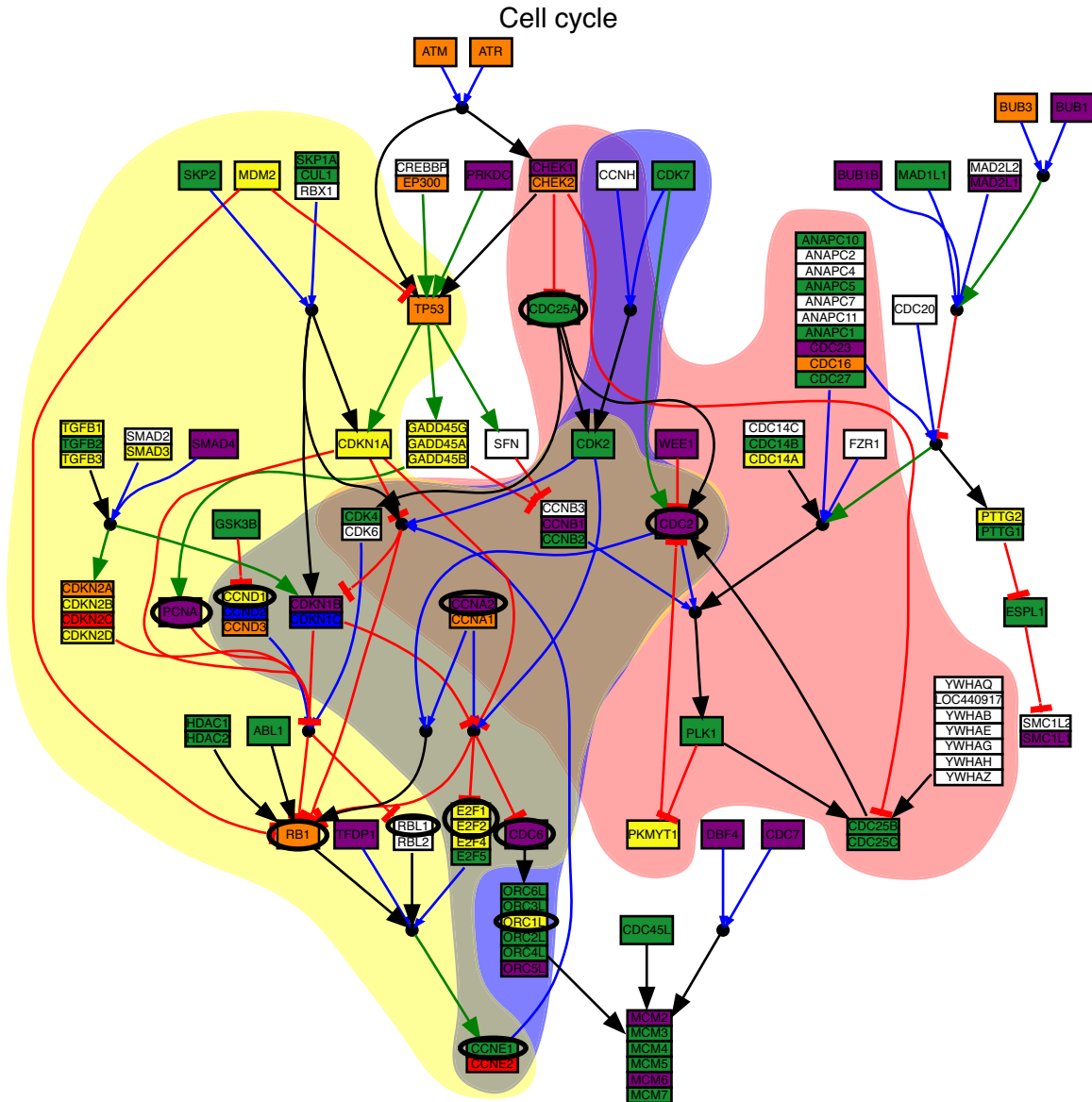


Figure 27. Cell cycle pathway (hsa04110) showing clustering, subpathways, and known E2F motifs. The large shaded areas indicate subpathways centered on core genes (CDK7:blue, WEE1:pink, HDAC7:yellow) as shown in Figure 25. The genes are colored based on a K-means clustering of their expression co-variation using a QPACA optimized subset of the ovarian data set. Circled genes are known targets of E2F (includes: CCNA2, CDC2, CDC6, CDC25A, CCND1, CCNE1, E2F1, E2F2, ORC1L, PCNA, RB1, RBL1). MAMF identified E2F motifs based on the whole collection of genes, either subnetwork, or the k-means cluster sets. The K-means motifs had the strongest match to known E2F motifs.

Figure 27 shows the distribution of the gene clusters in the pathway as well as the location of the subpathways defined in Section 6.4.1. This pathway contains 12 known

E2F target genes, of which 11 are represented in the data set. The purple and yellow clusters contain four; the green cluster has two, and the orange cluster has one. Within each cluster, the expression is highly coordinated, but the clusters do not co-vary with one another. This is consistent with the finding that a large number of other motifs co-occur with E2F and are likely to be biologically active.(Hon and Jain 2006) Specifically, AhR, NF-1, EGR, SMAD, CREB, TBP, AP-2, SP1, MYC, ETS, and other transcription factors have been identified as being functional within the same promoter as E2F.

Using MaMF to survey all of the promoters of the genes in the Cell Cycle pathway, E2F is the top scoring motif found (see Table 6). It was also found using each of the three subpathways as well as the expression module clusters. The strongest match to the known E2F consensus sequence was found when considering those gene sets with the largest number of genes with known E2F binding sites. Additionally, the expression modules did far better at finding the known E2F binding motifs than the subpathways, supporting the idea that these expression modules are, in fact, related to transcription factor activity. Finally, in addition to matching known E2F motifs, the MaMF generated motifs also matched other known TransFac motifs, many of which are known to co-occur with E2F. Specifically, motifs matching AhR, NF-1, SMAD, and Sp-1 were often present in the top 30 generated motifs.

The gene sets that resulted in particularly good E2F matches also resulted in a larger number of matches to these co-occurring motifs. Although this is just a single example, it is instructive. While E2F is ubiquitous (and largely annotated as such) within the different parts of the Cell Cycle pathway, the clusters that were identified by QPACA showed strong co-variation among *intra*-cluster genes but not among *inter*-cluster genes.

A number of high-scoring motifs were identified that co-occurred with E2F within these clusters, but a large number of high-scoring motifs did not have an annotated known transcription factor associate with them within TRANSFAC. In Hon and Jain's original paper, a similar feature was noted, where it was shown that the high-scoring motifs were likely to be biologically important due to underlying coherent expression effects. Here, a different approach will be used, making use of the pancreatic cancer cell line data.

Table 6. Selected MaMF motifs that are similar to E2F.

Gene set	Genes in set	Genes w/ known E2F BS	Total E2F matches	Motif similarity	Motif rank	Motif consensus	
all	94	11	55	0.85	2	<u>ttcGCGCCAAac</u>	
sub-pathways	CDK7	33	8	26	0.64	7	<u>gCGcCCGGAAArg</u>
	HDAC2	38	9	30	0.74	25	<u>CGsTccCGCCAccy</u>
	WEE1	35	3	17	0.66	5	<u>tgcCGGCCAAa.</u>
	1 red	2	0	7	0.78	28	<u>TCTCCCGCCAGG</u>
2 blue	2	0	7	0.73	28	<u>GTTCTCCCGCGCCAGs</u>	
clusters	3 yellow	21	4	23	0.81	25	<u>TtCGGCCaAgm</u>
	4 green	35	2	30	0.78	11	<u>TTCrCGcgAAAst</u>
	5 orange	12	1	20	0.68	11	<u>ggcgCGGGAAAAct.G</u>
	6 purple	22	4	31	0.83	1	<u>TTCgCGcgAAA.</u>
						ttCgCGCCAAac	

Gene sets were derived from the Cell Cycle pathway and the ovarian data set (see Table 5 for composition). The discrepancy between the number of genes listed for each set in this table and in previous tables is due to mapping between Entrez Gene IDs and Refseq IDs. Genes with known E2F binding sites were derived from the TransFac database (version 7.0 - public). The motif shown is the best scoring motif in the top 30 motifs found by MaMF. Note that those gene sets with larger numbers of known E2F genes also had higher scoring E2F motifs found. Underlined sequence is the 11 bp motif found by MaMF; additional sequence was obtained from the surrounding sequence. Capitalized nucleotides are highly conserved (>50% of the contributing sequences contain that nucleotide at that position). The motif in bold at the bottom is the annotated E2F consensus sequence.

7.3.2. *Permutation analysis of MaMF generated motifs in pancreatic cancer cell lines*

A more rigorous statistical analysis was performed using the pancreatic data set in order to explore two questions generated by the above analysis: (1) did the high scoring motifs found by MaMF using expression modules occur simply by chance and (2) did the

use of QPACA generated sample subsets affect the performance of the expression modules (would expression modules generated from the entire data set have a harder time finding relevant motifs).

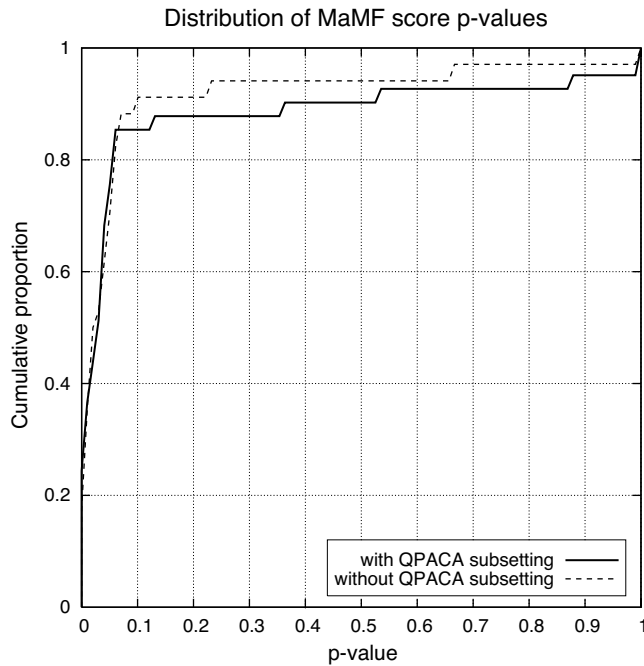


Figure 28. Distribution of MaMF score p-values for all expression modules. Using expression modules, MaMF is able to find better motifs than those found using random gene sets. The QPACA subsetting algorithm, however, does not seem to enhance MaMF's performance.

Given that a large number of the high-scoring motifs that MaMF found lacked any annotated known TF, it is natural to question whether motifs with competitive scores would have arisen by chance. To assess this probability, a null distribution of motifs was generated by MaMF using 100 random gene sets of the same size as the expression modules from the pancreatic cell line data set. The p-value for the motifs of a given module is the percentage of MaMF runs on random gene sets that resulted in higher scores than the corresponding expression module. Using this statistical measure, 75% of the expression modules were found to have significantly higher scores than random (see Figure 28). That is, the actual expression modules within the pancreatic cell line data set were strongly enriched for high-scoring motifs as detected by MaMF. In this particular

case, it did *not* appear that the QPACA subsetting made a significant impact on MaMF's ability to find high scoring motifs within expression modules identified without subsetting, but this is of less consequence. QPACA's subsetting algorithm is particularly powerful in fairly heterogeneous data sets because it is able to tease out signal from the noise of the uninvolved samples. In this case, all of the samples were related and this may have created a strong enough expression signal without a need for subsetting.

The key finding is that based on the enrichment for high-scoring motifs from the pathway expression modules, it seems likely that some significant fraction of the motifs that were found actually have biological significance. This argument can be strengthened by considering the fact that the pathways that were chosen for the pancreatic cancer analysis were enriched for genes known to contain E2F and NF-kappaB binding sites. Therefore, it is possible to compare which motifs were found for different pathways. Those pathways that were enriched for genes with E2F binding sites resulted in more E2F-like motifs than the pathways enriched for NF-kappaB genes and vice versa. So, analysis of the promoters of genes within the expression modules yielded both known TF motifs *and* high-scoring motifs of unknown provenance. Based on the enrichment of high-scoring motifs versus random gene sets, it appears that some interaction among the known TFs within these pathways as well as a significant number of additional, non-annotated TFs is responsible for some of the signal that QPACA is able to make use of in pathway recognition and augmentation tasks.

This preliminary data supports the idea that expression modules are related to transcription factor activity and that the involvement of transcription factors in gene expression may be useful in predicting pathway structure. Defining expression modules

was a bottleneck in this analysis, and a more automated approach to module selection would enhance future work in this direction. It is clear from the subpathway and cluster analyses that, just as not all samples are necessarily involved in an expression signature, not all genes are involved either. Thus, one possible approach to this issue is a modification of the QPACA subsetting algorithm to include gene subsetting as well as sample subsetting. This simultaneous optimization of gene set and sample set would identify more strongly coordinated transcriptional modules than studied here as the basis for further exploration.

Chapter 8

Conclusion and future directions

This work has described both the development and application of QPACA, a tool for pathway analysis. QPACA incorporates several facets of pathway analysis: pathway visualization and definition, visual analysis of experimental data, and computational pathway recognition and pathway membership prediction.

Chapter 1 and Chapter 2 introduced both the importance of pathway analysis for the elucidation of systems biology as well as current research efforts in this field.

Pathway analysis is still expanding with many competing and sometimes overlapping pathway and data storage systems as well as many variations on algorithms to augment pathway membership and structure. While exchange standards such as BioPAX are emerging in an effort to unify the disparate pathway databases, these efforts are still not robust enough to cover all pathway types and instances.

Chapter 3 described the development of QPACA, covering the pathway model, the optimization algorithm at the heart of QPACA's analytical methods, and the system architecture. QPACA's pathway representation was designed to be flexible, extensible and accessible. It is integrated into the Magellan data analysis application for easy use by biologists and can import pathway data in three separate formats: BioPAX, KEGG, and

the QPACA language. QPACA's utility in providing visual analysis of experimental data was demonstrated in Chapter 4 using several different types of human cancer data.

Chapter 5 and Chapter 6 addressed two steps in pathway structure elucidation using gene expression data. First, given a gene set postulated to be part of a pathway or coordinated process, QPACA demonstrated the ability to discriminate between real pathways and non-pathways. Second, given a gene set known to represent part of a pathway, QPACA was then able to significantly enrich for pathway genes over non-pathway genes in making predictions of putative pathway members. The first problem was termed pathway recognition and the second pathway augmentation. Specifically, for the pathway recognition problem, in 61% of cases known pathways (83% for larger pathways), QPACA was able to answer affirmatively while rejecting random pathways over 95% of the time. For the pathway augmentation problem, QPACA was able to yield enrichments for predicted pathway genes over random genes at rates of 2-fold or better the majority of the time, with rates of 10-fold or better 10–20% of the time depending on the background used. While enrichment rates on the level of 2-fold may or may not be of practical significance, it is likely that rates of 10-fold or better may help prioritize experimental exploration of candidate genes for elaborating pathways of particular biological interest. Additionally, Chapter 6 also introduced the idea that gene sets are recognized as pathways not because all genes within them are cross-correlated, but rather because individual genes form highly coordinated expression modules. By employing an automated method for identifying relevant sample subsets, QPACA has proved to be able to handle complex pathways relevant to major disease processes in humans.

There are several major areas for future work. First, the recognition and prediction algorithms can be extended to make use of new scoring metrics and optimization strategies. Given that the first, very simple, scoring metric yielded positive results, a more refined scoring metric would possibly be able to tease more information out of the data. The refined scoring metric presented at the end of Chapter 6 to deal with excessive prediction score ties is a simple example of this type of modification. Other metrics, which would allow both direction and magnitude to be taken into account in determining the score, can be considered that move beyond the simple correlation metric presented here. In particular, metrics that include pathway structure information could be particularly fruitful (e.g. by making use of graph-based algorithms that identify minimum spanning trees, maximum flow networks, etc.). Yet more challenging problems, including recognition and prediction of interactions (edges in the pathway graph) and full pathway induction (as opposed to just membership augmentation) are also areas for further research.

Chapter 7 explored the synergy that could arise by combining sequence-based analytical methods (e.g. motif finding in gene promoter sets) with the information derived from measurements of variation in gene expression or protein levels. The existence of the highly coordinated expression modules observed in Chapter 6 appears to stem, in part, from a set of common transcription factors that control the genes' regulation. Though many of the motif patterns that appear to be important within expression submodules of the pathways are not similar to known transcription factors.

This work has developed representational machinery for encoding knowledge about the biological networks that researchers conceptualize as “pathways.” The

computational approaches developed for performing analyses and making predictions about the composition and structure of these pathways have yielded an unexpected result: that it is frequently possible, in a human cellular system, to detect aspects of pathway structure from patterns of variation in gene expression. The signal from these variations manifests as small groups of genes among which expression is strongly coordinated (within some set of experimental conditions or samples), where many of these groups comprise a pathway. There appears to be little relationship, however, between the membership of these transcriptional modules and physical interactions of the sort that are typically depicted in curated pathway representations. It appears that the influence of transcription factors may explain a significant fraction of this puzzle, but the degree of multi-factorial control through transcription factor binding motifs in human cell biology may be more complex than has been contemplated. By combining data analysis, pathway knowledge, expression data, and sequence data it should be possible to tease out more pieces of the puzzle.

References

- Akutsu, T., S. Miyano, et al. (2000). "Algorithms for identifying Boolean networks and related biological networks based on matrix multiplication and fingerprint function." *J. Comput. Biol.* **7**(3-4): 331-43.
- Arkin, A., J. Ross, et al. (1998). "Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected Escherichia coli cells." *Genetics* **149**(4): 1633-48.
- Arkin, A., P. Shen, et al. (1997). "A Test Case of Correlation Metric Construction of a Reaction Pathway from Measurements." *Science* **277**(5330): 1275-1279.
- Bader, G. D., M. P. Cary, et al. (2006). "Pathguide: a pathway resource list." *Nucleic Acids Res* **34**(Database issue): D504-6.
- Becker, W., Y. Weber, et al. (1998). "Sequence characteristics, subcellular localization, and substrate specificity of DYRK-related kinases, a novel family of dual specificity protein kinases." *J Biol Chem* **273**(40): 25893-902.
- Berchuck, A., E. S. Iversen, et al. (2005). "Patterns of gene expression that characterize long-term survival in advanced stage serous ovarian cancers." *Clin Cancer Res* **11**(10): 3686-96.
- Bertagnolo, V., M. Marchisio, et al. (1998). "Nuclear association of tyrosine-phosphorylated Vav to phospholipase C-gamma 1 and phosphoinositide 3-kinase during granulocytic differentiation of HL-60 cells." *FEBS Lett* **441**(3): 480-4.
- D'Haeseleer, P., X. Wen, et al. (1999). "Linear modeling of mRNA expression levels during CNS development and injury." *Pac. Symp. Biocomput.*: 41-52.
- Dahlquist, K. D., N. Salomonis, et al. (2002). "GenMAPP, a new tool for viewing and analyzing microarray data on biological pathways." *Nat Genet* **31**(1): 19-20.
- Das, B., X. Shu, et al. (2000). "Control of intramolecular interactions between the pleckstrin homology and Dbl homology domains of Vav and Sos1 regulates Rac binding." *J Biol Chem* **275**(20): 15074-81.
- Delcommenne, M., C. Tan, et al. (1998). "Phosphoinositide-3-OH kinase-dependent regulation of glycogen synthase kinase 3 and protein kinase B/AKT by the integrin-linked kinase." *Proc Natl Acad Sci U S A* **95**(19): 11211-6.
- Elnitski, L., V. X. Jin, et al. (2006). "Locating mammalian transcription factor binding sites: a survey of computational and experimental techniques." *Genome Res* **16**(12): 1455-64.
- Friedman, N., M. Linial, et al. (2000). "Using Bayesian networks to analyze expression data." *J. Comput. Biol.* **7**(3-4): 601-20.

- Gulbins, E., K. Schlottmann, et al. (1995). "Molecular analysis of Ras activation by tyrosine phosphorylated Vav." Biochem Biophys Res Commun **217**(3): 876-85.
- Gysin, S., P. Rickert, et al. (2005). "Analysis of genomic DNA alterations and mRNA expression patterns in a panel of human pancreatic cancer cell lines." Genes Chromosomes Cancer **44**(1): 37-51.
- Hall, M., S. Bates, et al. (1995). "Evidence for different modes of action of cyclin-dependent kinase inhibitors: p15 and p16 bind to kinases, p21 and p27 bind to cyclins." Oncogene **11**(8): 1581-8.
- Hartemink, A. J., D. K. Gifford, et al. (2001). "Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks." Pac. Symp. Biocomput.: 422-33.
- Hobert, O., J. W. Schilling, et al. (1996). "SH3 domain-dependent interaction of the proto-oncogene product Vav with the focal contact protein zyxin." Oncogene **12**(7): 1577-81.
- Hon, L. S. and A. N. Jain (2006). "A deterministic motif finding algorithm with application to the human genome." Bioinformatics **22**(9): 1047-54.
- Hughes, T. R., M. J. Marton, et al. (2000). "Functional discovery via a compendium of expression profiles." Cell **102**(1): 109-26.
- Humbert, S., E. A. Bryson, et al. (2002). "The IGF-1/Akt pathway is neuroprotective in Huntington's disease and involves Huntingtin phosphorylation by Akt." Dev Cell **2**(6): 831-7.
- Ideker, T., V. Thorsson, et al. (2001). "Integrated genomic and proteomic analyses of a systematically perturbed metabolic network." Science **292**(5518): 929-34.
- Ihmels, J., G. Friedlander, et al. (2002). "Revealing modular organization in the yeast transcriptional network." Nat Genet **31**(4): 370-7.
- Jain, A. N., K. Chin, et al. (2001). "Quantitative analysis of chromosomal CGH in human breast tumors associates copy number abnormalities with p53 status and patient survival." Proc Natl Acad Sci U S A **98**(14): 7952-7.
- Jansen, R., H. Yu, et al. (2003). "A Bayesian networks approach for predicting protein-protein interactions from genomic data." Science **302**(5644): 449-53.
- Kanehisa, M. (1997). "A database for post-genome analysis." Trends Genet **13**(9): 375-6.
- Kanehisa, M. and S. Goto (2000). "KEGG: kyoto encyclopedia of genes and genomes." Nucleic Acids Res **28**(1): 27-30.
- Kanehisa, M., S. Goto, et al. (2004). "The KEGG resource for deciphering the genome." Nucleic Acids Res **32**(Database issue): D277-80.
- Karp, P. D., M. Riley, et al. (2002). "The EcoCyc Database." Nucleic Acids Res **30**(1): 56-8.
- Kitano, H. (2002). "Systems biology: a brief overview." Science **295**(5560): 1662-4.
- Kohn, K. W. (1999). "Molecular interaction map of the mammalian cell cycle control and DNA repair systems." Mol. Biol. Cell **10**(8): 2703-34.
- Liang, S., S. Fuhrman, et al. (1998). "Reveal, a general reverse engineering algorithm for inference of genetic network architectures." Pac. Symp. Biocomput.: 18-29.
- Liu, Y. F., D. Dorow, et al. (2000). "Activation of MLK2-mediated signaling cascades by polyglutamine-expanded huntingtin." J Biol Chem **275**(25): 19035-40.

- Matys, V., O. V. Kel-Margoulis, et al. (2006). "TRANSFAC and its module TRANSCompel: transcriptional gene regulation in eukaryotes." Nucleic Acids Res **34**(Database issue): D108-10.
- Moore, S. L., L. M. Selfors, et al. (2000). "Vav family proteins couple to diverse cell surface receptors." Mol Cell Biol **20**(17): 6364-73.
- Nagata, K., A. Puls, et al. (1998). "The MAP kinase kinase kinase MLK2 co-localizes with activated JNK along microtubules and associates with kinesin superfamily motor KIF3." Embo J **17**(1): 149-58.
- Nakao, M., H. Bono, et al. (1999). "Genome-scale Gene Expression Analysis and Pathway Reconstruction in KEGG." Genome Inform Ser Workshop Genome Inform **10**: 94-103.
- Neve, R. M., K. Chin, et al. (2006). "A collection of breast cancer cell lines for the study of functionally distinct cancer subtypes." Cancer Cell **10**(6): 515-27.
- Novak, B. A. and A. N. Jain (2006). "Pathway recognition and augmentation by computational analysis of microarray expression data." Bioinformatics **22**(2): 233-41.
- Ohtoshi, A., T. Maeda, et al. (2000). "Human p53(CDC)/Cdc20 associates with cyclin A and is phosphorylated by the cyclin A-Cdk2 complex." Biochem Biophys Res Commun **268**(2): 530-4.
- Olshen, A. B. and A. N. Jain (2002). "Deriving quantitative conclusions from microarray expression data." Bioinformatics **18**(7): 961-70.
- Paley, S. M. and P. D. Karp (2002). "Evaluation of computational metabolic-pathway predictions for Helicobacter pylori." Bioinformatics **18**(5): 715-24.
- Pinkel, D., R. Segraves, et al. (1998). "High resolution analysis of DNA copy number variation using comparative genomic hybridization to microarrays." Nat Genet **20**(2): 207-11.
- Segal, M. R., K. D. Dahlquist, et al. (2003). "Regression approaches for microarray data analysis." J Comput Biol **10**(6): 961-80.
- Shannon, P., A. Markiel, et al. (2003). "Cytoscape: a software environment for integrated models of biomolecular interaction networks." Genome Res **13**(11): 2498-504.
- Smith, V. A., E. D. Jarvis, et al. (2003). "Influence of network topology and data collection on network inference." Pac. Symp. Biocomput.: 164-75.
- Snijders, A. M., M. E. Nowee, et al. (2003). "Genome-wide-array-based comparative genomic hybridization reveals genetic homogeneity and frequent copy number increases encompassing CCNE1 in fallopian tube carcinoma." Oncogene **22**(27): 4281-6.
- Staunton, J. E., D. K. Slonim, et al. (2001). "Chemosensitivity prediction by transcriptional profiling." Proc Natl Acad Sci U S A **98**(19): 10787-92.
- Storey, J. D. and R. Tibshirani (2003). "Statistical significance for genomewide studies." Proc Natl Acad Sci U S A **100**(16): 9440-5.
- Stuart, J. M., E. Segal, et al. (2003). "A gene-coexpression network for global discovery of conserved genetic modules." Science **302**(5643): 249-55.
- Tusher, V. G., R. Tibshirani, et al. (2001). "Significance analysis of microarrays applied to the ionizing radiation response." Proc Natl Acad Sci U S A **98**(9): 5116-21.

- Veltman, J. A., J. Fridlyand, et al. (2003). "Array-based comparative genomic hybridization for genome-wide screening of DNA copy number in bladder tumors." Cancer Res **63**(11): 2872-80.
- Watson, D. K., L. Robinson, et al. (1997). "FLI1 and EWS-FLI1 function as ternary complex factors and ELK1 and SAP1a function as ternary and quaternary complex factors on the Egr1 promoter serum response elements." Oncogene **14**(2): 213-21.
- Weaver, D. C., C. T. Workman, et al. (1999). "Modeling regulatory networks with weight matrices." Pac. Symp. Biocomput.: 112-23.
- Weinmann, A. S., P. S. Yan, et al. (2002). "Isolating human transcription factor targets by coupling chromatin immunoprecipitation and CpG island microarray analysis." Genes Dev **16**(2): 235-44.
- Westfall, P. H. and S. S. Young (1993). Resampling-based Multiple Testing, Wiley, New York.
- Yamashita, R., Y. Suzuki, et al. (2006). "DBTSS: DataBase of Human Transcription Start Sites, progress report 2006." Nucleic Acids Res **34**(Database issue): D86-9.
- Yang, E., J. Zha, et al. (1995). "Bad, a heterodimeric partner for Bcl-XL and Bcl-2, displaces Bax and promotes cell death." Cell **80**(2): 285-91.
- Zhang, B. H., V. Ho, et al. (2001). "Specific involvement of G(alpha)2 with epidermal growth factor receptor signaling in rat hepatocytes, and the inhibitory effect of chronic ethanol." Biochem Pharmacol **61**(8): 1021-7.
- Zhang, D., K. Li, et al. (2005). "DYRK gene structure and erythroid-restricted features of DYRK3 gene expression." Genomics **85**(1): 117-30.

Appendix: Documentation of Code and Data

This section documents the use of the QPACA pathway language as well as the QPACA analytical methods initially referred to in Chapter 3 and later described in more detail in Chapter 5 and Chapter 6. This appendix should allow the reader to replicate the primary experiments and understand the software API. For specifics, such as data files and source code, the reader should consult the accompanying DVD.

Appendix A.

QPACA Model

A.1.1. QPACA Pathway Language

The pathway language is composed of 3 sections: a) pathway specific information, such as the name and any other pathway attributes b) element information, and c) edge information.

```

pathway_name = "RTK Pathway Sample"

#pathway elements
pathway_elements {
  e10[type=geneProduct;name=ELK1;GeneID=2002]
  e11.a[type=geneProduct;name=CCND1;GeneID=595]
  e12.a[type=geneProduct;name=CDK4;GeneID=1019]
  #complex element type
  a11[type=complex;objects=e11.a,e12.a]
  e13[type=geneProduct;name=RB1;GeneID=5925]
  e14.a[type=geneProduct;name=E2F1;GeneID=1869]
  e14.b[type=geneProduct;name=E2F2;GeneID=1870]
  e14.c[type=geneProduct;name=E2F3;GeneID=1871]
  e14.d[type=geneProduct;name=E2F4;GeneID=1874]
  a14[type=family;objects=e14.a,e14.b,e14.c,e14.d]
  e15.a[type=geneProduct;name=CCNE1;GeneID=898]
  e15.b[type=geneProduct;name=CCNA2;GeneID=890]
  #family element type
  a15[type=family;objects=e15.a,e15.b]
  e16[type=geneProduct;name=CDK2;GeneID=1017]
  e17[type=geneProduct;name=p14ARF;GeneID=1029]
  e18[type=geneProduct;name=MDM2;GeneID=4193]
  e19[type=geneProduct;name=CDKN2A;GeneID=1029;alt.name=p16]
  e20[type=geneProduct;name=CDKN1A;GeneID=1026;alt.name=p21]
  e21[type=geneProduct;name=TP53;GeneID=7157]
  #process element type
  e47[type=process;name=S-phase]
}
#pathway description
pathway {
  e10 -> e19 [weight=1]
  e10 -> a11 [weight=1]
  e19 -| a11 [weight=1]
  e10 -> e20 [weight=1]
  e20 -| a11 [weight=1]
  e20 -| e16 [weight=1]
  e13 -| a14 [weight=1]
  a14 -> a15 [weight=1]
  a15 -> e16 [weight=1]
  e16 -> e47 [weight=1]
  a14 -> e17 [weight=1]
  e17 -| e18 [weight=1]
  e18 -| e21 [weight=1]
  e21 -> e20 [weight=1]
  a11 -| e13 [weight=1]
}

```

Figure 29. Sample pathway description in QPACA Pathway Language.

Figure 29 shows a sample pathway description illustrating. All attributes are described using key-value pairs. Each pair is located on a separate line with an "=" separating the key from the value. Keys cannot contain any white space and values must

be delimited with quotation marks if they contain white space. Comment lines start with a "#".

The first section consists of a set of attribute values describing the pathway. The only required attribute is "pathway_name".

The "pathway_element" section defines all the components of the pathway. Each element must have a type and a unique ID. The ID is listed before the bracket. The allowed types are: geneProduct, family, complex, smallMolecule, and process. Elements of the "geneProduct" type must also have a GeneID, which denotes the Entrez Gene ID for that element. The "name" attribute governs the display name for the generated graph. Elements that group other elements (type "family" or "complex") must be defined after their constituent parts have been defined. "Family" denotes a group of elements which may all play the same role at a particular node in the graph. "Complex" denotes a group of elements that form a physical interaction and act together in the pathway.

The "pathway" section contains the interaction information for the pathway. The element IDs defined in the previous section are connected with symbols denoting edge type. Figure 29 shows the activation ("->") and inhibition ("-|") types. A neutral type ("--") is also currently supported.

A.1.2. Sample QPACA GUI Usage

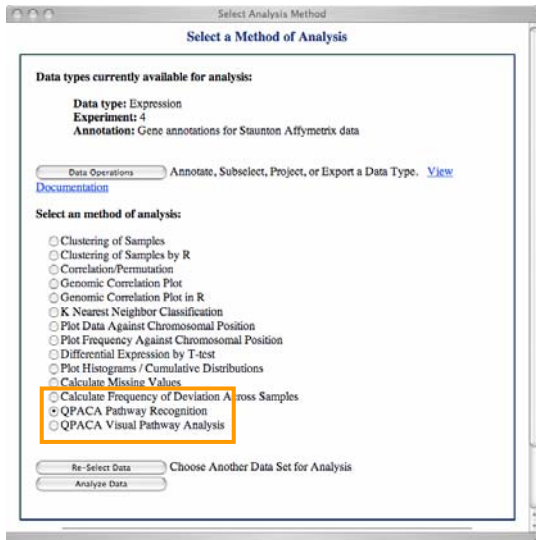


Figure 30. Analysis method selection in Magellan.

The Analysis Method Selection page in Magellan is the entry point for the QPACA analyses.

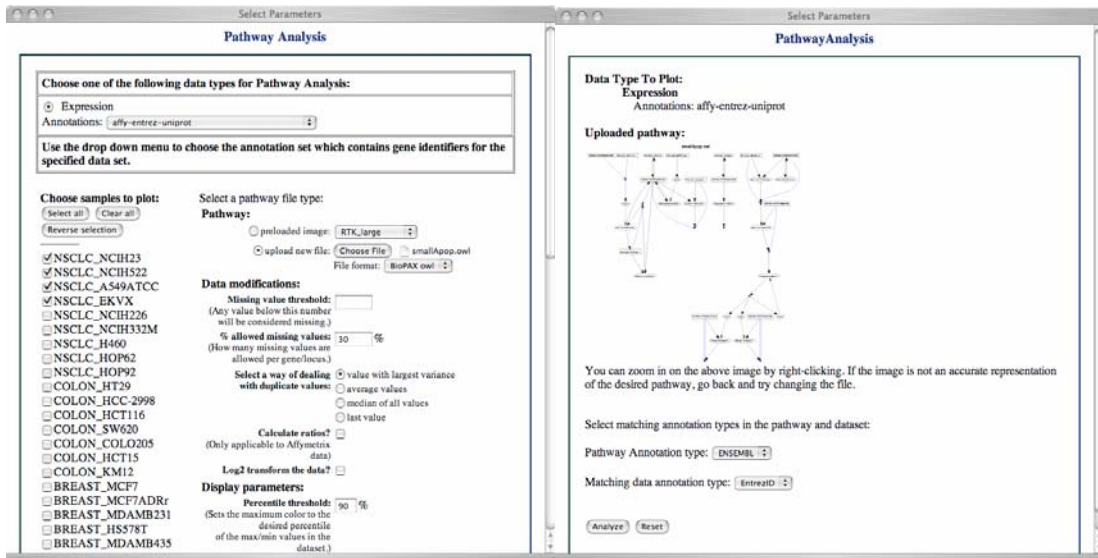


Figure 31. QPACA parameter selection

These pages present the user with a set of analysis parameters. On the first page, the top and left are Magellan specific parameters for data selection, while the column contains all of the QPACA parameters. The next page shows the automatically generated image for verification purposes and allows the user to choose the annotation types that match between the data in the Magellan database and the objects in the pathway.

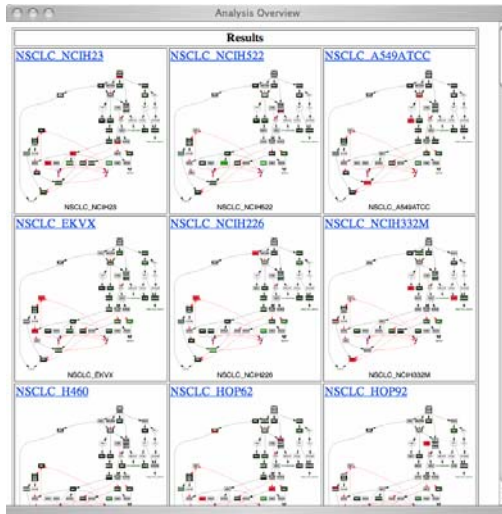


Figure 32. QPACA data visualization results page.

This page shows visualization result page with a separate image for each selected sample in the data set. QPACA can also generate statistical summary images (such as correlations to a specified gene or F-statistic based on sample groups). The images are in SVG format and each one links to a larger version. The larger versions (also in SVG format) contain embedded links to the Entrez Gene database. The computational analysis results page (not shown) is presented as a simple directory listing of output files.

Appendix B.

QPACA Analytical Module

B.1.1. PathwayAnalysis Usage

PathwayAnalysis runs the QPACA pathway recognition algorithm. It is necessary to run this prior to running PredictGenes. Data files should be tab-delimited text files with gene IDs in the first column and sample names (optional) across the top. edu.ucsf.qpaca.pathway.PathwayData generates appropriate pathway subsets data files given a pathway description file and a data set, though any file of the appropriate format can be used.

```
> PathwayAnalysis [option] ... datafile
```

Options:

-opt <n>	sets the number of optimization steps to n (default = 600)
-rgenes	perform random gene set permutation
-rdata	perform randomized data permutation
-inner <n>	sets the number of starting points to n (default = 20)
-outer <n>	sets the number of permutations to n (default = 1000)
-subsetsize <n>	sets the size of the subset of samples to be chosen to n (default = 20)
-header	if the datafile has a header line (containing sample names)

-datasize <n>	sets the number of genes in the data file to n (defaults to the number of data rows in the file)
-numsamples <n>	sets the number of total samples in the datafile to n (defaults to the number of data columns in the file)
-outputdir <directory_name>	output directory
-outfile <filename>	base name for output file
-help	this information

B.1.2. PredictGenes Usage

PredictGenes is responsible for identifying potential new pathway members. The sample file for PredictGenes is derived from the PathwayAnalysis output.

```
> PredictGenes [-options] -samplefile samplefilename
      -datafile datafilename -pathdatafile pathwaygenesfilename
```

Options:

-nsamples <n>	number of samples in the sample subset
-ntotalsamples	perform random gene set permutation
-outputdir <directory_name>	output directory
-outfile <filename>	base name for output file
-help	this information

The following three tab-delimited files are necessary:

samplefile	specifies which samples of the data set to use in predicting new pathway genes.
datafile	should contain the data for those genes which are not known to belong to the pathway
pathdatafile	contains the data for the genes known to belong to the pathway

Appendix C.

Code Documentation

C.1.1. PathwayAnalysis Usage.....	101
C.1.2. QPACA Visual Tools Code Documentation.....	128

QPACA Analytical Tools Code Documentation

PathwayAnalysis	pathway recognition module
PredictGenes	pathway membership augmentation module
PathwayUtil	a collection of utility functions for the PathwayAnalysis and PredictGenes modules

PathwayAnalysis

PathwayAnalysis is QPACA's pathway recognition module. The output consists of a final score and a final, optimized subset of samples that were used to calculate that score.

Usage:

> PathwayAnalysis [options] ... datafile

Options	
-opt <i>n</i>	sets the number of optimization steps to <i>n</i> (default = 600)
-rgenes	perform random gene set permutation
-rdata	perform randomized data permutation
-starts <i>n</i>	sets the number of starting points to <i>n</i> (default = 20)
-perm <i>n</i>	sets the number of permutations to <i>n</i> (default = 1000)
-subsetSize <i>n</i>	sets the size of the subset of samples to be chosen to <i>n</i> (default = 20)
-header	if the datafile has a header line (containing sample names)
-datasize <i>n</i>	sets the number of genes in the data file to <i>n</i> (defaults to the number of data rows in the file)

-numsamples <i>n</i>	sets the number of total samples in the datafile to <i>n</i> (defaults to the number of data columns in the file)
-corrout	print individual correlation values (useful for debugging)
-intOut <i>n</i>	determines which scores to print out. If $n == 0$, only reports the best overall score across all starting points, otherwise reports all scores (default).
-outputdir <i>directory_name</i>	output directory
-outfile <i>filename</i>	base name for output file
-selectgenes <i>n</i>	subselects <i>n</i> genes after subselecting samples
-help	print usage information

Functions

compute_correlation

Computes all correlations in a 2 dimensional array.

fisher_yates_shuffle

Shuffles an array of size *n* in place.

get_random_subset

Chooses a random sample subset.

optimize

Performs optimization of samples.

optimizeWithGenes

Performs optimization of genes.

print_help

Prints out a short description of how to run the program.

random_int

Generates a random integer in $[0,n)$.

random_subset

Chooses a random subset without replacement.

compute_correlation

Computes all correlations in a 2 dimensional array.

```
void compute_correlation(  
    int colsize,  
    int rowsize,  
    int *cols,  
    int *rows,  
    double *correlations,  
    double **data);
```

Parameter Description

colsize
number of columns in the dataset

rowsize
number of rows in the dataset

cols
the columns to use in computing the correlation

rows
the rows to use in computing the correlation

correlations
stores all of the correlation values

data
a 2-dimensional array containing the data

fisher_yates_shuffle

Shuffles an array of size n in place.

```
void fisher_yates_shuffle(  
    double *array,
```

```
int n) ;
```

Parameter Description

array	the array to shuffle
n	the size of the array

get_random_subset

Chooses a random sample subset.

```
void get_random_subset (  
    int k,  
    int n,  
    int *set,  
    int *rest) ;
```

Parameter Description

k	subset size
n	total number of samples
set	int array holding the indices of the random subset
rest	int array holding the rest of the sample indices

Discussion

Generates a random sample subset of size k from a total sample set of size n and stores the subset in two arrays

optimize

Performs optimization of samples.

```
double optimize(  
    int maxSteps,  
    int subsetsize,  
    int totalsize,  
    int datasize,  
    int corsize,  
    int *genes,  
    double *corr,  
    double **data,  
    int *subset);
```

Parameter Description

maxSteps	maximum number of optimization steps
subsetsize	size of the subset to optimize for
totalsize	total number of columns in the dataset (samples)
datasize	total number of rows in the dataset (genes)
corsize	size of the output correlation array
genes	array of which genes (indices) to optimize with
corr	output correlation array
data	2-dimensional array of data
subset	final sample subset

function result
the final score

optimizeWithGenes

Performs optimization of genes.

```
double optimizeWithGenes(  
    int maxSteps,  
    int nGenes,  
    int nSamples,  
    int nTotalSamples,  
    int nTotalGenes,  
    double **data,  
    int *sampleSubset,  
    int *geneSubset);
```

Parameter Description

maxSteps
maximum number of optimization steps

subsetSize
size of the subset to optimize for

nTotalSamples
total number of columns in the dataset (samples)

nTotalGenes
total number of rows in the dataset (genes)

corsize
size of the output correlation array

data
2-dimensional array of data

sampleSubset
array of which samples (indices) to optimize with

geneSubset

final sample subset

function result
the final score

print_help

Prints out a short description of how to run the program.

```
void print_help(  
    void) ;
```

random_int

Generates a random integer in [0,n).

```
int random_int (  
    int n) ;
```

Parameter Description

n
the upper bound

function result
a random integer in [0,n)

random_subset

Chooses a random subset without replacement.

```
void random_subset (  
    int k ,  
    int n ,  
    int *arr ) ;
```

Parameter Description

k	size of the subset
n	total size of the set
arr	array to store the chosen subset

#defines

FILESEP

The character used to separate parts of a filestring.

HEADER

default is no header

NUMBEROPTIMIZATIONS

default value of n for -opt option

NUMBERPERMUTATIONS

default value of n for -perm option

NUMBERSTARTINGPOINTS

default value of n for -starts options

NUMGENES

default value of n for -selectgenes option

SELECTGENES

default is to not perform gene subselection

SUBSETSIZE

default value of n for -subsetsize option

FILESEP

The character used to separate parts of a filestring.

```
/*Unix*/  
#define FILESEP "/";
```

Discussion

When compiling for Unix-based systems use "/"; for Windows use "\\\".

HEADER

default is no header

```
#define HEADER 0
```

NUMBEROPTIMIZATIONS

default value of n for -opt option

```
#define NUMBEROPTIMIZATIONS 600
```

NUMBERPERMUTATIONS

default value of n for -perm option

```
#define NUMBERPERMUTATIONS 1000
```

NUMBERSTARTINGPOINTS

default value of n for -starts option

```
#define NUMBERSTARTINGPOINTS 20
```

NUMGENES

default value of n for -selectgenes option

```
#define NUMGENES 20
```

SELECTGENES

default is to not perform gene subselection

```
#define SELECTGENES 0
```

SUBSETSIZE

default value of n for -subsetsize option

```
#define SUBSETSIZE 20
```

PredictGenes

PredictGenes is QPACA's pathway membership augmentation module.

Prior to running PredictGenes, it is necessary to ascertain pathway recognition using PathwayAnalysis. The sample subset and final score provided by PathwayAnalysis are then input into PredictGenes along with a list of non-pathway genes. PredictGenes assigns scores to the query genes based on the optimized sample subset and score from PathwayAnalysis.

Usage:

```
> PredictGenes [-options] -samplefile samplefilename -datafile  
datafilename -pathdatafile pathwaygenesfilename
```

Options	
-nsamples <i>n</i>	number of samples in the sample subset
-ntotalsamples <i>n</i>	number total samples in the data set
-outfile <i>filename</i>	desired name for the output file (default: root of the datafilename)
-outputdir <i>directory_name</i>	output directory (default: current directory)
-help	prints usage information

The following three tab-delimited files are necessary:

samplefile	specifies which samples of the data set to use in predicting new pathway genes.
datafile	should contain the data for those genes which are not known to belong to the pathway.
pathdatafile	contains the data for the genes known to belong to pathway in question

Functions

compute_correlation

Computes all correlations in a 2-dimensional array.

compute_original_score

Calculates the score for the known pathway gene subset.

compute_scores

Computes the new scores.

print_help

Prints usage information.

read_sample_file

Reads a sample file with no header.

compute_correlation

Computes all correlations in a 2-dimensional array.

```
void compute_correlation(  
    int colsize,  
    int rowsize,  
    int *cols,  
    double *correlations,  
    double **data);
```

Parameter Description

colsize

number of columns in the dataset

rowsize

number of rows in the dataset

cols

the columns to use in computing the correlation

correlations

stores all of the correlation values

data

a 2-dimensional array containing the data

compute_original_score

Calculates the score for the known pathway gene subset.

See Also: [compute_scores](#)

```
double compute_original_score(  
    int nsamples,  
    int ntotalsamples,  
    int nsubsetRows,  
    int ndataRows,  
    int *sampleList,  
    double **subsetData,  
    double **data);
```

Parameter Description

nsamples
the number of samples in the sample subset

ntotalsamples
the total number of samples

nsubsetRows
the number of rows in the known pathway gene subset

ndataRows
the number of total data rows

sampleList
the optimized sample subset

subsetData
the data for the known pathway gene subset

data
the data for the genes to be scored

function result

the score for the known pathway gene subset

compute_scores

Computes the new scores.

```
double* compute_scores (  
    int nsubsetCols,  
    int ndataCols,  
    int nsubsetRows,  
    int ndataRows,  
    int *sampleList,  
    double **subsetData,  
    double **data);
```

Parameter Description

nsubsetCols
the number of columns in the sample subset

ndataCols
the number of total data columns

nsubsetRows
the number of rows in the known pathway gene subset

ndataRows
the number of total data rows

sampleList
the optimized sample subset

subsetData
the data for the known pathway gene subset

data
the data for the genes to be scored

Discussion

The scoring function consists of the mean of the top 10% of all gene-gene correlations in the gene subset. This differs from the scoring function in PathwayAnalysis, which is the median of all gene-gene correlations.

print_help

Prints out a short description of how to run the program.

```
void print_help(  
    void);
```

read_sample_file

Reads a sample file with no header.

```
void read_sample_file(  
    FILE *fd,  
    int *samples,  
    int ncols);
```

Parameter Description

fd	the sample file
ncols	the number of columns in the file
samples	an array used to store the sample information

PathwayUtil

A collection of utility functions for the the PathwayAnalysis and PredictGenes modules.

Functions

bcoeff2

Calculates the binomial coefficient for n choose 2.

compare_doubles

Compares two doubles.

count_cols

Counts the columns in a tab-delimited file.

count_rows

Counts the rows in a file.

doublearray_copy

Copies an array of doubles.

intarray_copy

Copies an array of ints.

lvector

Allocates an unsigned long vector with subscript range v[nl..nh].

mean

Calculates the arithmetic mean of an array of doubles.

median

Finds the median of an array of doubles.

pearsons

Calculates the pearsons correlation coefficient between two arrays.

read_file_data

Reads a tab-delimited data file with no headers.

read_file_data_ids

Reads a tab-delimited data file with a header column.

read_file_with_headers

Reads a tab-delimited data file with a header column.

select_kth_smallest

Selects the k-th smallest value in a double array.

sort

Sorts an array arr[0..n-1] into ascending numerical order.

variance

Calculates the variance of an array.

bcoeff2

Calculates the binomial coefficient for n choose 2.

```
int bcoeff2(  
    int n) ;
```

Parameter Description

n

function result

the binomial coefficient

compare_doubles

Compares two doubles.

```
int compare_doubles(  
    const void *a,  
    const void *b) ;
```

Parameter Description

Parameter Description

a
b

function result

-1 if $a < b$, 1 if $a > b$, and 0 if they're equal

count_cols

Counts the columns in a tab-delimited file.

```
int count_cols(  
    FILE *fd);
```

Parameter Description

fd
the file pointer to the file

function result

the number of columns

count_rows

Counts the rows in a file.

```
int count_rows(  
    FILE *fd);
```

Parameter Description

fd
the file pointer to the file

function result
the number of rows

doublearray_copy

Copies an array of doubles.

```
void doublearray_copy(  
    int n,  
    double *src,  
    double *dest);
```

Parameter Description

n
the size of the array

src
the original array

dest
the new array

intarray_copy

Copies an array of ints.

```
void intarray_copy(  
    int n,  
    int *src,  
    int *dest);
```

```
int n,  
int *src,  
int *dest);
```

Parameter Description

n	the size of the array
src	the original array
dest	the new array

lvector

Allocates an unsigned long vector with subscript range v[nl..nh].

```
unsigned long *lvector(  
    long nl,  
    long nh);
```

Parameter Description

data	the array
n	the size of the array

function result
the variance

mean

Calculates the arithmetic mean of an array of doubles.

```
double mean(  
    unsigned long n,  
    double arr[]);
```

Parameter Description

n
the size of the array

arr
the array

function result
the arithmetic mean

median

Finds the median of an array of doubles.

```
double median(  
    unsigned long n,  
    double arr[]);
```

Parameter Description

n
the size of the array

arr
the array

function result
the median

pearsons

Calculates the pearsons correlation coefficient between two arrays.

```
double pearsons(  
    double *data1,  
    double *data2,  
    int n) ;
```

Parameter Description

data1	the first array
data2	the second array
n	the size of the arrays

function result
the pearsons correlation coefficient

read_file_data

Reads a tab-delimited data file with no headers.

```
void read_file_data(  
    FILE *fd,  
    double **data,  
    int ncols) ;
```

Parameter Description

fd
a file pointer to the file

data
a 2-dimensional array of doubles used to store the data

ncols
the number of columns in the file

Discussion

Stores the data in the file in a 2-dimensional array of doubles.

read_file_data_ids

Reads a tab-delimited data file with a header column containing IDs.

See Also: [read_file_data](#)

```
void read_file_data_ids(  
    FILE *fd,  
    double **data,  
    char **ids,  
    int ncols);
```

Parameter Description

fd
a file pointer to the file

data
a matrix of doubles used to store the data

ids
the string array used to store ids

ncols
the number of columns in the file

Discussion

Just as the `read_file_data` function, this function stores the data in the file in a 2-dimensional array of doubles. In addition, the first column is assumed to contain data ids which are stored in a separate string array.

`read_file_with_headers`

Reads a tab-delimited data file with both a header column containing IDs and a header row containing sample names.

See Also: [read_file_data](#), [read_file_data_ids](#)

```
void read_file_with_headers(  
    FILE *fd,  
    double **data,  
    char **samples,  
    char **ids,  
    int ncols);
```

Parameter Description

<code>fd</code>	a file pointer to the file
<code>data</code>	a matrix of doubles used to store the data
<code>samples</code>	a string array used to store sample names
<code>ids</code>	the string array used to store ids
<code>ncols</code>	the number of columns in the file

Discussion

Just as the `read_file_data` function, this function stores the data in the file in a 2-dimensional array of doubles. In addition, the first column is assumed to contain data ids which are stored in a separate string array. Also, the first row is assumed to contain sample names, which are stored in a separate string array.

select_kth_smallest

Selects the k-th smallest value in a double array.

```
double select_kth_smallest(  
    unsigned long k,  
    unsigned long n,  
    double arr[]);
```

Parameter Description

k	
n	the length of the array
arr	the array

function result
the k-th smallest value

Discussion

Returns the k-th smallest value in the array `arr[0..n-1]`. The input array will be rearranged to have this value in location `arr[k-2]`, with all smaller elements moved to `arr[0..k-2]` (in arbitrary order) and all large elements in `arr[k..n-1]` (also in arbitrary order). from "Numerical Recipes in C", pg. 342

sort

Sorts an array `arr[0..n-1]` into ascending numerical order.

```
void sort(  
    unsigned long n,  
    double arr[]);
```

Parameter Description

<code>n</code>	the size of the array
<code>arr</code>	the array to sort

Discussion

Uses the Quicksort algorithm and replaces the input array with its sorted rearrangement.

variance

Calculates the variance of an array.

```
double variance(  
    double *data,  
    int n);
```

Parameter Description

<code>data</code>	the array
<code>n</code>	the size of the array

function result
the variance

#defines

MAXLENGTH

Default length for allocating new strings.

SWAP

Swaps the location of two values.

MAXLENGTH

Default length for allocating new strings.

```
#define MAXLENGTH 3000
```

SWAP

Swaps the location of two values.

```
#define SWAP(a,b) temp=(a);(a)=(b);(b)=temp;
```

QPACA	
<u>edu.ucsf.qpaca</u>	QPACA (Quantitative Pathway Analysis in Cancer) provides the classes necessary to model pathways (<u>edu.ucsf.qpaca.pathway</u>) as well as the classes necessary for pathway input/output (<u>edu.ucsf.qpaca.parser</u>).

Pathway Modeling	
<u>edu.ucsf.qpaca.pathway</u>	Models a pathway and any associated data.
<u>edu.ucsf.qpaca.pathway.algorithm</u>	Implements several graph-based algorithms for pathway model navigation.

Pathway I/O	
<u>edu.ucsf.qpaca.parser</u>	Handles input/output of pathway information from the QPACA pathway model.
<u>edu.ucsf.qpaca.parser.util</u>	Supports the QPACA parser classes by defining language constants and providing parsing utility methods.

Package edu.ucsf.qpaca

QPACA (Quantitative Pathway Analysis in Cancer) provides the classes necessary to model pathways (edu.ucsf.qpaca.pathway) as well as the classes necessary for pathway input/output (edu.ucsf.qpaca.parser).

See:

[Description](#)

Class Summary	
FileUploadBean	A Java Bean that handles uploading files from a web page.
PathwayAnalysis	Runs the QPACA Pathway Analysis Tools within Magellan.

Package edu.ucsf.qpaca Description

QPACA (Quantitative Pathway Analysis in Cancer) provides the classes necessary to model pathways (edu.ucsf.qpaca.pathway) as well as the classes necessary for pathway input/output (edu.ucsf.qpaca.parser). This package includes two top-level classes used to implement the Magellan-based user interface. QPACA uses the following publicly available java libraries:

- OpenJGraph 0.9.2
- batik 1.6
- junit 3.8.1

Additionally, the user interface is dependent on edu.ucsf.Magellan and the Apache Tomcat java servlet container.

Package edu.ucsf.qpaca.pathway

Models a pathway and any associated data.

See:

[Description](#)

Interface Summary	
PathwayComponent	The base class for all pathway components, which specifies methods for getting and setting the various common attributes.

Class Summary	
Pathway	A biological pathway represented as a directed graph.
PathwayAssembly	An collection of objects in a pathway that can be found at a PathwayVertex.
PathwayData	Container for data values that are used to color pathways.
PathwayEdge	An edge in a pathway.
PathwayElement	An element of a pathway.
PathwayFactory	The factory for creating Vertices and Edges in a Pathway class.
PathwaySVG	Handles manipulation and coloring (by data value) of SVG files generated by GraphViz.
PathwayUtil	Static utility methods for manipulating data values.
PathwayVertex	A vertex in a pathway.

Package edu.ucsf.qpaca.pathway Description

Models a pathway and any associated data. The pathway model is based the on the OpenJGraph library, which provides classes for the creation and manipulation of graphs. At the root of the model is the PathwayComponent interface, which defines the base unit of each pathway model component. The model itself is composed of nodes (PathwayVertex) and edges (PathwayEdge) which belong to a pathway (Pathway). Each node can represent either a physical entity (PathwayAssembly or PathwayElement) or an event. Event nodes can also contain event modifier entities (either PathwayAssemblies or PathwayElements). Each PathwayAssembly is composed of one or many PathwayElements or other PathwayAssemblies. PathwayAssemblies can represent groups of PathwayElements that either act at the same node (but independently) or form a complex (with physical interactions).

In addition, this package also contains classes for manipulating data and pathway images (PathwayData and PathwaySVG). Finally, several classes (PathwayUtil, PathwayFactory, WeakPathway) include methods to facilitate use of the pathway model.

Package edu.ucsf.qpaca.pathway.algorithm

Implements several graph-based algorithms for pathway model navigation.

See:

[Description](#)

Class Summary	
PathwayMinimumSpanningTree	A concrete implementation of the minimum spanning tree algorithm using Kruskal's method specifically geared to Pathway objects.
ShortestPathAlgorithm	Abstract class for implementing the shortest path algorithm.
ShortestPathDijkstraAlgorithm	A concrete implementation of ShortestPathAlgorithm using Dijkstra's method.

Package edu.ucsf.qpaca.pathway.algorithm Description

Implements several graph-based algorithms for pathway model navigation.

Package edu.ucsf.qpaca.parser

Handles input/output of pathway information from the QPACA pathway model.

See:

[Description](#)

Class Summary	
MapBioPAXtoQPACA	Maps between BioPAX keywords and QPACA keywords.
MapKGMLtoQPACA	Maps between KEGG KGML keywords and QPACA keywords.
ParseQPACALanguage	Creates QPACA pathways from files written in the QPACA Pathway language.
PathwayToDot	Translates Pathways into GraphViz .dot files.
TestBioPaxUtil	Tests the BioPaxUtil Class.
TestKGMLParsing	Tests the KGML-QPACA mapper.
TestKGMLUtil	Tests the KGMLUtil Class.
TestParsing	Tests the BioPAX-QPACA mapper.
TestPathLang	Tests the QPACA language parser.

Package edu.ucsf.qpaca.parser Description

Handles input/output of pathway information from the QPACA pathway model. ParseQPACALanguage parses the QPACA file format. MapBioPAXtoQPACA and MapKGMLtoQPACA parse BioPAX and KGML formats, respectively. PathwayToDot creates dot format files for parsing by the Graphviz graph visualization program.

Package edu.ucsf.qpaca.parser.util

Supports the QPACA parser classes by defining language constants and providing parsing utility methods.

See:

[Description](#)

Class Summary	
BioPaxConstants	BioPAX Constants.
BioPaxUtil	BioPax Utility Class.
KGMLConstants	KGML Constants.
KGMLUtil	KEGG KGML Utility Class.
OwlConstants	OWL (Web Ontology Language) Constants.
QPACAConstants	QPACA Constants.
QPACAUtil	QPACA Pathway language Utility Class.
RdfConstants	RDF (Resource Description Framework) Constants.
RdfQuery	Enables XPath-"lite" Queries on RDF Documents.
RdfUtil	Misc RDF Utilities.

Exception Summary	
BioPaxException	Exception thrown to indicate a syntax error in a BioPAX file.
KGMLException	Exception thrown to indicate a syntax error in a KGML file.
QPACAEException	Exception thrown to indicate a syntax error in a QPACA file.

Package edu.ucsf.qpaca.parser.util Description

Supports the QPACA parser classes by defining language constants and providing parsing utility methods. Each file format throws it's own type of exception. Constants are defined in the Constants classes and utility methods in the Util classes. In addition, RdfConstants, RdfQuery, and RdfUtil enable more straightforward parsing of the XML-based formats.



edu.ucsf.qpaca

Class FileUploadBean

java.lang.Object

└─ edu.ucsf.qpaca.FileUploadBean

```
public class FileUploadBean
```

```
extends java.lang.Object
```

A Java Bean that handles uploading files from a web page.

Constructor Summary

[FileUploadBean\(\)](#)

Method Summary

void	doUpload (javax.servlet.http.HttpServletRequest request) Uploads the file specified on the HTML input page.
java. lang. String []	getAllFieldValues (java.lang.String fieldName) Returns an array of String objects containing all of the values the given parameter has, or null if the parameter does not exist.
java. lang. String	getContentType () Retrieves the content type of the uploaded file.
java. util. Map	getFields () Retrieves the HTML input form's input elements.
java. lang. String	getFieldValue (java.lang.String fieldName) Retrieves the value of an HTML input element
java. lang. String	getFilename () Retrieves the filename of the uploaded file.

java. lang. String	getFilepath() Retrieves the complete filepath on the client side of the uploaded file.
void	setSavePath (java.lang.String savePath) Sets the location that the file should be stored in on the server.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

FileUploadBean

```
public FileUploadBean()
```

Method Detail

doUpload

```
public void doUpload(javax.servlet.http.HttpServletRequest request)
    throws java.io.IOException
```

Uploads the file specified on the HTML input page. First extracts the field names and values from the HTML form. Then extracts the uploaded file and saves it to the directory specified by the savePath and assigns the file's name, path, contentType to the filename, filepath and contentType fields.

Parameters:

request - the HTTP request that is needed to process the extracted HTML forms's element names-values and the uploaded file.

Throws:

java.io.IOException

getAllFieldValues

```
public java.lang.String[] getAllFieldValues(java.lang.String fieldName)
```

Returns an array of String objects containing all of the values the given parameter has, or null if the parameter does not exist.

Parameters:

fieldName - name of the input element

Returns:

values of the input element

getContent-type

```
public java.lang.String getContent-type()
```

Retrieves the content type of the uploaded file.

Returns:

the content type

getFields

```
public java.util.Map getFields()
```

Retrieves the HTML input form's input elements.

Returns:

the input elements

getFieldValue

```
public java.lang.String getFieldValue(java.lang.String fieldName)
```

Retrieves the value of an HTML input element

Parameters:

fieldName - name of the input element

Returns:

value of the input element

getFilename

```
public java.lang.String getFilename()
```

Retrieves the filename of the uploaded file.

Returns:
the filename

getFilepath

```
public java.lang.String getFilepath()
```

Retrieves the complete filepath on the client side of the uploaded file.

Returns:
the filepath

setSavePath

```
public void setSavePath(java.lang.String savePath)
```

Sets the location that the file should be stored in on the server.

Parameters:
savePath - server file location

edu.ucsf.qpaca

Class PathwayAnalysis

java.lang.Object

└─ edu.ucsf.Magellan.AnalysisThread

└─ edu.ucsf.qpaca.PathwayAnalysis

All Implemented Interfaces:

java.lang.Runnable

```
public class PathwayAnalysis
```

```
extends edu.ucsf.Magellan.AnalysisThread
```

Runs the QPACA Pathway Analysis Tools within Magellan. Handles creation of output directories and manages any errors generated during analysis.

Field Summary

Fields inherited from class edu.ucsf.Magellan.AnalysisThread

Analysis, application, running, sessionID

Constructor Summary

[PathwayAnalysis](#)(edu.ucsf.Magellan.AnalysisInfo analysis, java.lang.String sessionID, javax.servlet.ServletContext application, javax.servlet.ServletRequest request, [Pathway](#) path)

Initializes a new PathwayAnalysis thread given the provided parameters

[PathwayAnalysis](#)(edu.ucsf.Magellan.AnalysisInfo analysis, java.lang.String sessionID, java.lang.String filepath, java.lang.String annottype, java.lang.String pathAnnotType, [Pathway](#) path)

Initializes a new PathwayAnalysis thread given the provided parameters

Method Summary

boolean	deleteDir (java.io.File dir) Deletes all files and subdirectories under dir.
void	run () Runs the analysis.
void	writeLog (java.io.File log, java.lang.String analysisType, edu.ucsf.Magellan.DataType dt) Writes out a log file detailing all of the analysis parameters.

Methods inherited from class edu.ucsf.Magellan.AnalysisThread
isRunning, setRunning

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PathwayAnalysis

```
public PathwayAnalysis(edu.ucsf.Magellan.AnalysisInfo analysis,
    java.lang.String sessionID,
    javax.servlet.ServletContext application,
    javax.servlet.HttpServletRequest request,
    Pathway path)
```

Initializes a new PathwayAnalysis thread given the provided parameters

Parameters:

analysis - the AnalysisInfo object storing the analysis parameters
 sessionID - the session identifier
 application - the ServletContext for this instance of PathwayAnalysis
 request - the client request information for this instance of PathwayAnalysis
 path - the pathway to analyze

PathwayAnalysis

```
public PathwayAnalysis(edu.ucsf.Magellan.AnalysisInfo analysis,
    java.lang.String sessionID,
```

```
java.lang.String filepath,  
java.lang.String annottype,  
java.lang.String pathAnnotType,  
Pathway path)
```

Initializes a new PathwayAnalysis thread given the provided parameters

Parameters:

analysis - the AnalysisInfo object storing the analysis parameters
sessionID - the session identifier
filepath - the data file path
annottype - the annotation type for the data
pathAnnotType - the matching annotation type in the pathway
path - the pathway to analyze

Method Detail

deleteDir

```
public boolean deleteDir(java.io.File dir)
```

Deletes all files and subdirectories under dir.

Parameters:

dir - the directory to delete

Returns:

true if all deletions successful, false otherwise.

run

```
public void run()
```

Runs the analysis.

Specified by:

run in interface java.lang.Runnable

Overrides:

run in class edu.ucsf.Magellan.AnalysisThread

writeLog

```
public void writeLog(java.io.File log,  
                    java.lang.String analysisType,  
                    edu.ucsf.Magellan.DataType dt)  
    throws java.io.IOException
```

Writes out a log file detailing all of the analysis parameters.

Parameters:

log - the file
analysisType - the type of analysis performed
dt - the data type used in the analysis

Throws:

java.io.IOException - If any problems with file access occur.

edu.ucsf.qpaca.pathway

Interface PathwayComponent

All Known Implementing Classes:

[Pathway](#), [PathwayAssembly](#), [PathwayEdge](#), [PathwayElement](#), [PathwayVertex](#)

```
public interface PathwayComponent
```

The base class for all pathway components, which specifies methods for getting and setting the various common attributes.

Method Summary	
java. lang. Object	getAttribute (java.lang.String type) Returns the attribute mapped to the specified attribute type.
java. util.Map	getAttributes () Returns the attributes of this Element.
java. lang. String	getID () Returns the ID of this PathwayComponent.
boolean	hasAttribute (java.lang.String type) Tests to see if this PathwayComponent has this type of attribute.
java. lang. Object	removeAttribute (java.lang.String key) Removes a specific attribute.
void	setAttribute (java.lang.String key, java.lang.Object value) Creates a new attribute.

Method Detail

getAttribute

```
java.lang.Object getAttribute(java.lang.String type)
```

Returns the attribute mapped to the specified attribute type.

Parameters:

type - the attribute type

Returns:

the attribute specified by the type

getAttributes

java.util.Map **getAttributes**()

Returns the attributes of this Element.

Returns:

the attributes

getID

java.lang.String **getID**()

Returns the ID of this PathwayComponent.

Returns:

the ID

hasAttribute

boolean **hasAttribute**(java.lang.String type)

Tests to see if this PathwayComponent has this type of attribute.

Parameters:

type - the attribute type

Returns:

true if this type of attribute exists;

removeAttribute

java.lang.Object **removeAttribute**(java.lang.String key)

Removes a specific attribute.

Parameters:

key - the attribute type

setAttribute

void **setAttribute**(java.lang.String key,
java.lang.Object value)

Creates a new attribute.

Parameters:

key - the attribute type

value - the attribute

edu.ucsf.qpaca.pathway

Class Pathway

```
java.lang.Object
├── salvo.jesus.graph.GraphImpl
│   ├── salvo.jesus.graphDirectedGraphImpl
│   │   └── edu.ucsf.qpaca.pathway.Pathway
```

All Implemented Interfaces:

[PathwayComponent](#), java.io.Serializable, salvo.jesus.graphDirectedGraph, salvo.jesus.graph.Graph, salvo.jesus.graph.WeightedGraph

```
public class Pathway
```

```
extends salvo.jesus.graphDirectedGraphImpl
implements salvo.jesus.graph.WeightedGraph, PathwayComponent, java.io.Serializable
```

A biological pathway represented as a directed graph.

See Also:

[Serialized Form](#)

Field Summary

protected java. util. HashMap	attributes A HashMap of attributes.
--	--

Fields inherited from class salvo.jesus.graph.GraphImpl

factory, traversal

Constructor Summary

[Pathway](#)()
Constructs a new, empty pathway.

Method Summary

void	addEdge (PathwayEdge pe) Adds a previously created edge.
------	--

salvo.jesus.graph.WeightedEdge	addEdge (salvo.jesus.graph.Vertex v1, salvo.jesus.graph.Vertex v2, double weight) Convenience method to add a WeightedEdge with a specified weight into the WeightedGraph.
PathwayEdge	addEdge (salvo.jesus.graph.Vertex v1, salvo.jesus.graph.Vertex v2, int newType) Adds a new edge connecting vertex v1 and vertex v2.
PathwayEdge	addEdge (salvo.jesus.graph.Vertex source, salvo.jesus.graph.Vertex sink, int newType, double newWeight) Adds a PathwayEdge with a specific weight to the Pathway.
java.util.Set	getAssemblies (java.util.Set vertices) Retrieves all Assemblies represented by the specified Vertices
java.lang.Object	getAttribute (java.lang.String type) Returns the attribute mapped to the specified attribute type.
java.util.Map	getAttributes () Returns the attributes of this Pathway.
salvo.jesus.graph.Vertex	getClosest (salvo.jesus.graph.Vertex v) Determines the Vertex that is 'closest' to the Vertex specified.
PathwayElement	getElement (java.lang.String input, java.lang.String inputType) Retrieves the PathwayElement represented by the input:inputType
java.util.Set	getElements () Retrieves all PathwayElements in this pathway
java.util.Set	getElements (java.util.Set vertices) Retrieves all PathwayElements represented by the specified Vertices.
java.lang.String	getID () Returns the unique ID of this PathwayComponent.
java.util.List	getPath (salvo.jesus.graph.Vertex v1, salvo.jesus.graph.Vertex v2) Retrieves the shortest path from Vertex v1 to Vertex v2.
PathwayVertex	getVertex (java.lang.String input, java.lang.String inputType) Retrieves the Vertex containing the PathwayElement represented by the input:inputType
PathwayVertex	getVertexByID (java.lang.String id) Retrieves the Vertex with the specified ID.
java.util.Set	getVertexSet (PathwayElement elem) Retrieves all Vertices containing the specified PathwayElement.
java.util.Set	getVertexSet (java.lang.String input, java.lang.String inputType) Retrieves all Vertices containing the PathwayElement represented by the input:inputType
boolean	hasAttribute (java.lang.String type) Tests to see if this PathwayComponent has the specified type of attribute.
salvo.jesus.graph.WeightedGraph	minimumSpanningTree () Determine a minimum spanning tree for the weighted graph.
java.util.Set	nConnected (salvo.jesus.graph.Vertex startVertex, int n, int dir) Retrieves all vertices that are within n steps from the starting Vertex
int	pathEffect (java.util.List path) Retrieves the overall effect of the specified path based on PathwayEdge types.

java.lang. Object	removeAttribute (java.lang.String key) Removes a specific attribute
void	setAttribute (java.lang.String key, java.lang.Object value) Creates a new attribute
void	setMinimumSpanningTreeAlgorithm (PathwayMinimumSpanningTree algo) Sets the algorithm used to determine the minimum spanning tree.
void	setShortestPathAlgorithm (ShortestPathAlgorithm algo) Sets the algorithm used to determine the shortest path spanning tree.
salvo.jesus. graph. WeightedGraph	shortestPath (salvo.jesus.graph.Vertex vertex) Determine a shortest path spanning tree for the weighted graph.

Methods inherited from class salvo.jesus.graph.DirectedGraphImpl

getEdge, getIncomingAdjacentVertices, getIncomingEdges,
getOutgoingAdjacentVertices, getOutgoingEdges, isCycle, isPath

Methods inherited from class salvo.jesus.graph.GraphImpl

add, addEdge, addEdge, addGraphAddEdgeListener, addGraphAddVertexListener,
addGraphRemoveEdgeListener, addGraphRemoveVertexListener, addListener,
cloneVertices, containsEdge, containsVertex, forgetConnectedSets,
getAdjacentVertices, getAdjacentVertices, getConnectedSet, getConnectedSet,
getDegree, getDegree, getEdges, getEdgesCount, getEdgeSet, getGraphFactory,
getTraversal, getVertexSet, getVertices, getVerticesCount, getVerticesIterator,
isConnected, remove, removeEdge, removeEdges, removeGraphAddEdgeListener,
removeGraphAddVertexListener, removeGraphRemoveEdgeListener,
removeGraphRemoveVertexListener, removeListener, setGraphFactory, setTraversal,
toString, traverse

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface salvo.jesus.graph.Graph

add, addEdge, addEdge, addGraphAddEdgeListener, addGraphAddVertexListener,
addGraphRemoveEdgeListener, addGraphRemoveVertexListener, addListener,
cloneVertices, getAdjacentVertices, getAdjacentVertices, getConnectedSet,
getConnectedSet, getDegree, getDegree, getEdges, getEdgesCount, getEdgeSet,
getGraphFactory, getTraversal, getVertexSet, getVertices, getVerticesCount,
getVerticesIterator, isConnected, remove, removeEdge, removeEdges,
removeGraphAddEdgeListener, removeGraphAddVertexListener,
removeGraphRemoveEdgeListener, removeGraphRemoveVertexListener, removeListener,
setGraphFactory, setTraversal, traverse

Field Detail

attributes

protected java.util.HashMap **attributes**

A HashMap of attributes.

Constructor Detail

Pathway

```
public Pathway()
```

Constructs a new, empty pathway.

Method Detail

addEdge

```
public void addEdge(PathwayEdge pe)
    throws java.lang.Exception
```

Adds a previously created edge. Calls `DirectedGraphImpl.addEdge(edge)`.

Parameters:

pe - the edge to add

Throws:

java.lang.Exception

addEdge

```
public salvo.jesus.graph.WeightedEdge addEdge(salvo.jesus.graph.Vertex v1,
    salvo.jesus.graph.Vertex v2,
    double weight)
    throws java.lang.Exception
```

Convenience method to add a `WeightedEdge` with a specified weight into the `WeightedGraph`. The default `addEdge(v1, v2)` will add a `WeightedEdge` with `weight=1.0`, after which you can call `setWeight()` to specify the weight.

Specified by:

`addEdge` in interface `salvo.jesus.graph.WeightedGraph`

Returns:

the new `WeightedEdge`

Throws:

java.lang.Exception

addEdge

```
public PathwayEdge addEdge(salvo.jesus.graph.Vertex v1,  
                             salvo.jesus.graph.Vertex v2,  
                             int newType)  
    throws java.lang.Exception
```

Adds a new edge connecting vertex v1 and vertex v2. Calls `DirectedGraphImpl.addEdge(v1, v2)` and sets the type of the edge to `newType` and the weight to 1.0.

Parameters:

v1 - Vertex that will be the source of the PathwayEdge
v2 - Vertex that will be the sink of the PathwayEdge
newType - the type of the new PathwayEdge

Returns:

the new PathwayEdge

Throws:

java.lang.Exception

addEdge

```
public PathwayEdge addEdge(salvo.jesus.graph.Vertex source,  
                             salvo.jesus.graph.Vertex sink,  
                             int newType,  
                             double newWeight)  
    throws java.lang.Exception
```

Adds a PathwayEdge with a specific weight to the Pathway. Calls `DirectedGraphImpl.addEdge(v1, v2)` and sets the type of the edge to `newType` and the weight to `newWeight`.

Parameters:

source - Vertex that will be the source of the Edge
sink - Vertex that will be the sink of the Edge
newType - the type of the Edge
newWeight - the weight of the Edge

Returns:

the new PathwayEdge

Throws:

java.lang.Exception

getAssemblies

```
public java.util.Set getAssemblies(java.util.Set vertices)
```

Retrieves all Assemblies represented by the specified Vertices

Parameters:

vertices - a Set of vertices

Returns:

a List of PathwayElements

getAttribute

```
public java.lang.Object getAttribute(java.lang.String type)
```

Returns the attribute mapped to the specified attribute type.

Specified by:

[getAttribute](#) in interface [PathwayComponent](#)

Parameters:

type - the attribute type

Returns:

the attribute specified by the type

getAttributes

```
public java.util.Map getAttributes()
```

Returns the attributes of this Pathway.

Specified by:

[getAttributes](#) in interface [PathwayComponent](#)

Returns:

the attributes

getClosest

```
public salvo.jesus.graph.Vertex getClosest(salvo.jesus.graph.Vertex v)
```

Determines the Vertex that is 'closest' to the Vertex specified. The definition of the closest vertex in this context is a vertex that is directly adjacent to Vertex v where the edge has the least weight.

Specified by:

[getClosest](#) in interface [salvo.jesus.graph.WeightedGraph](#)

Parameters:

v - the starting vertex

Returns:

the Vertex closes to v.

getElement

```
public PathwayElement getElement(java.lang.String input,
```

java.lang.String inputType)

Retrieves the PathwayElement represented by the input:inputType

Parameters:

input - the input value
inputType - the input type

Returns:

a PathwayVertex, or null if no such Vertex found.

getElements

```
public java.util.Set getElements()
```

Retrieves all PathwayElements in this pathway

Returns:

a Set of all PathwayElements

getElements

```
public java.util.Set getElements(java.util.Set vertices)
```

Retrieves all PathwayElements represented by the specified Vertices.

Parameters:

vertices - a Set of vertices

Returns:

a List of PathwayElements

getID

```
public java.lang.String getID()
```

Returns the unique ID of this PathwayComponent. Since pathways do not have IDs, returns null

Specified by:

[getID](#) in interface [PathwayComponent](#)

Returns:

null

getPath

```
public java.util.List getPath(salvo.jesus.graph.Vertex v1,  
                               salvo.jesus.graph.Vertex v2)
```

Retrieves the shortest path from Vertex v1 to Vertex v2. Uses the shortest path spanning tree determined by shortestpath.

Parameters:

v1 - start Vertex

v2 - end Vertex

Returns:

a list of the edges of the shortest path from v1 to v2

getVertex

```
public PathwayVertex getVertex(java.lang.String input,  
                                   java.lang.String inputType)
```

Retrieves the Vertex containing the PathwayElement represented by the input:inputType

Parameters:

input - the input value

inputType - the input type

Returns:

a PathwayVertex, or null if no such Vertex found

getVertexByID

```
public PathwayVertex getVertexByID(java.lang.String id)
```

Retrieves the Vertex with the specified ID.

Parameters:

id - the ID of the desired Vertex

Returns:

the PathwayVertex with the specified ID

getVertexSet

```
public java.util.Set getVertexSet(PathwayElement elem)
```

Retrieves all Vertices containing the specified PathwayElement.

Parameters:

elem - the input PathwayElement

Returns:

a PathwayVertex

getVertexSet

```
public java.util.Set getVertexSet(java.lang.String input,  
                                   java.lang.String inputType)
```

Retrieves all Vertices containing the PathwayElement represented by the input:inputType

Parameters:

input - the input value

inputType - the input type

Returns:

a PathwayVertex

hasAttribute

```
public boolean hasAttribute(java.lang.String type)
```

Tests to see if this PathwayComponent has the specified type of attribute.

Specified by:

[hasAttribute](#) in interface [PathwayComponent](#)

Parameters:

type - the attribute type

Returns:

true if this type of attribute exists

minimumSpanningTree

```
public salvo.jesus.graph.WeightedGraph minimumSpanningTree()
```

Determine a minimum spanning tree for the weighted graph. There is no guarantee that the same method call will result in the same result, as long as it satisfies the property of a minimum spanning tree.

Specified by:

minimumSpanningTree in interface salvo.jesus.graph.WeightedGraph

Returns:

Subgraph connecting all the Vertices such that the sum of the weights of the Edges is at least as small as the sum of the weights of any other collection of Edges connecting all the Vertices.

nConnected

```
public java.util.Set nConnected(salvo.jesus.graph.Vertex startVertex,  
                                int n,  
                                int dir)
```

Retrieves all vertices that are within n steps from the starting Vertex

Parameters:

startVertex - the start Vertex

n - number of steps from the start Vertex

dir - the direction to search (1=downstream, -1=upstream, 0=both)

Returns:

a Set of vertices that are n-connected to the start Vertex

pathEffect

```
public int pathEffect(java.util.List path)
```

Retrieves the overall effect of the specified path based on PathwayEdge types.

Parameters:

path - a list of Pathway edges defining a path

Returns:

1 if the overall effect is positive, -1 if negative

removeAttribute

```
public java.lang.Object removeAttribute(java.lang.String key)
```

Removes a specific attribute

Specified by:

[removeAttribute](#) in interface [PathwayComponent](#)

Parameters:

key - the attribute type

setAttribute

```
public void setAttribute(java.lang.String key,  
                          java.lang.Object value)
```


Creates a new attribute

Specified by:

[setAttribute](#) in interface [PathwayComponent](#)

Parameters:

key - the attribute type

value - the attribute

setMinimumSpanningTreeAlgorithm

```
public void setMinimumSpanningTreeAlgorithm(PathwayMinimumSpanningTree algo)
```

Sets the algorithm used to determine the minimum spanning tree.

setShortestPathAlgorithm

```
public void setShortestPathAlgorithm(ShortestPathAlgorithm algo)
```

Sets the algorithm used to determine the shortest path spanning tree.

shortestPath

```
public salvo.jesus.graph.WeightedGraph shortestPath(salvo.jesus.graph.Vertex vertex)
```

Determine a shortest path spanning tree for the weighted graph. Shortest path spanning tree need not be unique. Therefore, there is no guarantee that calling this method twice for the same weighted graph will return exactly the same shortest path spanning tree, unless there is only one shortest path spanning tree.

Also note that the graph returned by this method is a new instance of `WeightedGraph`. However, its vertices and edges will be the same instance as those of this `WeightedGraph`. Therefore, **do not** modify the contents of the returned `WeightedGraph` such that any of its vertices or edges are removed.

Specified by:

`shortestPath` in interface `salvo.jesus.graph.WeightedGraph`

Parameters:

vertex - The Vertex in the weighted graph that we want to get the shortest paths to all other vertices.

Returns:

Shortest spanning subgraph from the vertex parameter to all other vertices that are in the same connected set as the vertex.

edu.ucsf.qpaca.pathway

Class PathwayAssembly

```
java.lang.Object
├── java.util.AbstractCollection<E>
│   ├── java.util.AbstractList<E>
│   │   └── java.util.ArrayList
│   │       └── edu.ucsf.qpaca.pathway.PathwayAssembly
```

All Implemented Interfaces:

[PathwayComponent](#), java.io.Serializable, java.lang.Cloneable, java.lang.Comparable, java.lang.Iterable, java.util.Collection, java.util.List, java.util.RandomAccess

```
public class PathwayAssembly
```

extends java.util.ArrayList

implements java.io.Serializable, [PathwayComponent](#), java.lang.Comparable

An collection of objects in a pathway that can be found at a PathwayVertex.

See Also:

[Serialized Form](#)

Field Summary	
protected java. util. HashMap	attributes A hashtable of attributes.
protected java. lang. String	id The unique id of the PathwayAssembly.

protected java. lang. String	type The type of PathwayAssembly.
---------------------------------------	--

Fields inherited from class java.util.AbstractList
modCount

Constructor Summary
PathwayAssembly () Constructs a new, empty PathwayAssembly.
PathwayAssembly (java.lang.String newType, java.lang.String newID) Constructs a new PathwayAssembly object of the specified type.
PathwayAssembly (java.lang.String newType, java.lang.String newID, java.util.List l) Constructs a new PathwayAssembly object of the specified type with the specified list of PathwayElements.
PathwayAssembly (java.lang.String newType, java.lang.String newID, java.util.List l, java.util.Map newAttributes) Constructs a new PathwayAssembly object of the specified type with the specified list of PathwayElements and attributes.

Method Summary	
int	compareTo (java.lang.Object o) Compares this PathwayAssembly to another Object.
java. lang. Object	getAttribute (java.lang.String type) Returns the attribute mapped to the specified attribute type.
java. util.Map	getAttributes () Returns the attributes of this PathwayAssembly.
java. lang. String	getID () Returns the unique ID.
java. lang. String	getType () Returns the type.
boolean	hasAttribute (java.lang.String type) Tests to see if this PathwayComponent has this particular type of attribute.
java. lang. Object	removeAttribute (java.lang.String key) Removes a specific attribute.

void	setAttribute (java.lang.String key, java.lang.Object value) Creates a new attribute.
void	setType (java.lang.String newType) Sets the type.
java. lang. String	toString () Returns a String representation of this PathwayAssembly.

Methods inherited from class java.util.ArrayList

add, add, addAll, addAll, clear, clone, contains, ensureCapacity, get, indexOf, isEmpty, lastIndexOf, remove, remove, removeRange, set, size, toArray, toArray, trimToSize

Methods inherited from class java.util.AbstractList

equals, hashCode, iterator, listIterator, listIterator, subList

Methods inherited from class java.util.AbstractCollection

containsAll, removeAll, retainAll

Methods inherited from class java.lang.Object

finalize, getClass, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.util.List

containsAll, equals, hashCode, iterator, listIterator, listIterator, removeAll, retainAll, subList

Field Detail

attributes

protected java.util.HashMap **attributes**

A hashtable of attributes.

id

protected java.lang.String **id**

The unique id of the PathwayAssembly.

type

protected java.lang.String **type**

The type of PathwayAssembly. Indicates whether it's a set of aliases or a compound.

Constructor Detail

PathwayAssembly

```
public PathwayAssembly()
```

Constructs a new, empty PathwayAssembly.

PathwayAssembly

```
public PathwayAssembly(java.lang.String newType,  
                        java.lang.String newID)
```

Constructs a new PathwayAssembly object of the specified type.

Parameters:

`newType` - the type of this PathwayAssembly
`newID` - this PathwayAssembly's ID

PathwayAssembly

```
public PathwayAssembly(java.lang.String newType,  
                        java.lang.String newID,  
                        java.util.List l)
```

Constructs a new PathwayAssembly object of the specified type with the specified list of PathwayElements.

Parameters:

`newType` - the type of this PathwayAssembly
`newID` - this PathwayAssembly's ID
`l` - a List of PathwayElements that this PathwayAssembly contains

PathwayAssembly

```
public PathwayAssembly( java.lang.String newType,  
                        java.lang.String newID,  
                        java.util.List l,  
                        java.util.Map newAttributes)
```

Constructs a new PathwayAssembly object of the specified type with the specified list of PathwayElements and attributes.

Parameters:

`newType` - the type of this PathwayAssembly
`newID` - this PathwayAssembly's ID
`l` - a List of PathwayElements that this PathwayAssembly contains
`newAttributes` - this PathwayAssembly's attributes

Method Detail

compareTo

```
public int compareTo( java.lang.Object o)
```

Compares this PathwayAssembly to another Object. If the Object is another PathwayAssembly, returns 0 if the two Assemblies have identical types and ids.

Specified by:

`compareTo` in interface `java.lang.Comparable`

Returns:

the value 0 if the argument is an equivalent PathwayAssembly

Throws:

`java.lang.ClassCastException` - if the argument is not a valid PathwayAssembly

getAttribute

```
public java.lang.Object getAttribute(java.lang.String type)
```

Returns the attribute mapped to the specified attribute type.

Specified by:

[getAttribute](#) in interface [PathwayComponent](#)

Parameters:

type - the attribute type

Returns:

the attribute specified by the type

getAttributes

```
public java.util.Map getAttributes()
```

Returns the attributes of this PathwayAssembly.

Specified by:

[getAttributes](#) in interface [PathwayComponent](#)

Returns:

the attributes

getID

```
public java.lang.String getID()
```

Returns the unique ID.

Specified by:

[getID](#) in interface [PathwayComponent](#)

Returns:

the type

getType

```
public java.lang.String getType()
```

Returns the type.

Returns:
the type

hasAttribute

```
public boolean hasAttribute(java.lang.String type)
```

Tests to see if this PathwayComponent has this particular type of attribute.

Specified by:
[hasAttribute](#) in interface [PathwayComponent](#)

Parameters:
type - the attribute type

Returns:
true if this type of attribute exists;

removeAttribute

```
public java.lang.Object removeAttribute(java.lang.String key)
```

Removes a specific attribute.

Specified by:
[removeAttribute](#) in interface [PathwayComponent](#)

Parameters:
key - the attribute type

setAttribute

```
public void setAttribute(java.lang.String key,
```


java.lang.Object value)

Creates a new attribute.

Specified by:

[setAttribute](#) in interface [PathwayComponent](#)

Parameters:

key - the attribute type

value - the attribute

setType

```
public void setType(java.lang.String newType)
```

Sets the type.

Parameters:

newType -

toString

```
public java.lang.String toString()
```

Returns a String representation of this PathwayAssembly.

Overrides:

toString in class java.util.AbstractCollection

Returns:

type.id and a list of objects contained in the Assembly

edu.ucsf.qpaca.pathway
Class PathwayData

java.lang.Object
└─ edu.ucsf.qpaca.pathway.PathwayData

```
public class PathwayData
```

```
extends java.lang.Object
```

Container for data values that are used to color pathways. Includes methods for some basic types of data manipulation.

Constructor Summary

[PathwayData](#)()

Creates an empty dataset.

[PathwayData](#)(java.io.File datafile, java.io.File annotFile, java.lang.String annotType, boolean logValues, double percentile, java.lang.String dupmethod, boolean calcRatios, double missingValueThreshold, double percAllowedMissing)

Creates a new PathwayData object using the specified dataset.

Method Summary

protected
double

[calculateColor](#)(double value)

Calculates the color value for a particular data value.

protected
double

[calculateColor](#)(double value, double dMax, double dMin, double dMid)

Calculates the color value for a particular data value given a defined data value range.

[Pathway](#)

[colorByCorrelation](#)([Pathway](#) path, java.lang.String pathAnnotType)

Colors a pathway based on the correlations.

Pathway	colorByPvalue (Pathway path, int pValueIndex, java.lang.String pathAnnotType, java.lang.String title)
Pathway	colorBySample (Pathway path, int sampleIndex, java.lang.String pathAnnotType) Colors a pathway based on the sample values for the given sample index.
Pathway	colorBySample (Pathway path, java.lang.String pathAnnotType) Colors a pathway based on the sample values for the first sample index.
int	createDataFiles (Pathway path, java.lang.String pathAnnotType, java.lang.String filepath, java.lang.String fileroot) Creates the pathway subset files for this dataset, given a pathway.
boolean	generateCorrelationData () Calculates the correlations of each gene to the gene with the specified correlationID.
java.util.Map	getCorrelationData ()
java.lang.String	getCorrelationID () Returns the annotation ID used to calculate correlations.
double	getDataMax () Returns the largest data value in this dataset.
double	getDataMin () Returns the smallest data value in this dataset.
java.util.List	getSamplenames () Returns the samplenames.
int	getSize () Returns the number of data values in this dataset.
boolean	isEmpty () Returns true if this dataset contains no values.
static void	main (java.lang.String[] args) This method allows for testing the class on the command line.
void	readDataFile (java.io.File datafile, java.io.File annotFile, java.lang.String annotType, boolean logValues, double maxPercentile, java.lang.String dupmethod, boolean calcRatios, double missingValueThreshold, double percAllowedMissing) Reads in a datafile.
boolean	setCorrelationID (java.lang.String id) Sets the annotation ID for which to calculate correlations.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PathwayData

```
public PathwayData()
```

Creates an empty dataset.

PathwayData

```
public PathwayData(java.io.File datafile,  
                  java.io.File annotFile,  
                  java.lang.String annotType,  
                  boolean logValues,  
                  double percentile,  
                  java.lang.String dupmethod,  
                  boolean calcRatios,  
                  double missingValueThreshold,  
                  double percAllowedMissing)  
    throws java.io.FileNotFoundException,  
           java.io.IOException
```

Creates a new PathwayData object using the specified dataset. The dataset must be contained in a datafile with the following characteristics:

- rows contain genes genes
- columns contain samples
- the first column contains gene identifiers
- the first row contains the sample names
- missing data is identified using the empty string or "NA"

Any additional gene annotations must be contained in a separate annotation file with the following characteristics:

- the first column contains gene identifiers
- additional columns contain the different gene annotations
- annotation types are listed in the first row

Parameters:

datafile - the data file
annotFile - the annotation file

annotType - the annotation type in the annotation file that corresponds to an annotation type in the Pathway.
logValues - true if the datafile contains log values, false otherwise
percentile - the percentile of the max/min threshold at which color is most intense
dupmethod - method used to resolve duplicates
calcRatios - if true, the values in the file need to have ratios calculated
missingValueThreshold - the minimum threshold for valid data values
percAllowedMissing - the percent of data values allowed to be missing for the gene to be considered not missing

Throws:

FileNotFoundException
java.io.IOException

Method Detail

calculateColor

```
protected double calculateColor(double value)
```

Calculates the color value for a particular data value.

Parameters:

value - the data value

Returns:

the color value

calculateColor

```
protected double calculateColor(double value,  
                                double dMax,  
                                double dMin,  
                                double dMid)
```

Calculates the color value for a particular data value given a defined data value range.

Parameters:

value - the data value

dMax - the largest data value

dMin - the smallest data value

dMid - the data value that represents 0

colorByCorrelation

```
public Pathway colorByCorrelation(Pathway path,  
                                   java.lang.String pathAnnotType)
```

Colors a pathway based on the correlations.

Parameters:

path - the pathway to color
pathAnnotType - the annotation type of the pathway elements

Returns:

the colored pathway

colorByPvalue

```
public Pathway colorByPvalue(Pathway path,  
                               int pValueIndex,  
                               java.lang.String pathAnnotType,  
                               java.lang.String title)
```

colorBySample

```
public Pathway colorBySample(Pathway path,  
                              int sampleIndex,  
                              java.lang.String pathAnnotType)
```

Colors a pathway based on the sample values for the given sample index.

Parameters:

path - the pathway to color
sampleIndex - the index of the sample to use for coloring
pathAnnotType - the annotation type of the pathway elements

Returns:

the colored pathway

colorBySample

```
public Pathway colorBySample(Pathway path,  
                             java.lang.String pathAnnotType)
```

Colors a pathway based on the sample values for the first sample index.

Parameters:

path - the pathway to color
pathAnnotType - the annotation type of the pathway elements

Returns:

the colored pathway

createDataFiles

```
public int createDataFiles(Pathway path,  
                           java.lang.String pathAnnotType,  
                           java.lang.String filepath,  
                           java.lang.String fileroot)  
    throws java.io.IOException
```

Creates the pathway subset files for this dataset, given a pathway.

Parameters:

path - the Pathway for which to create the datafiles

Returns:

the number of data rows in the pathway subset file

Throws:

java.io.IOException

generateCorrelationData

```
public boolean generateCorrelationData()
```

Calculates the correlations of each gene to the gene with the specified correlationID.

Returns:

true if the correlation ID exists, false otherwise

getCorrelationData

```
public java.util.Map getCorrelationData()
```

getCorrelationID

```
public java.lang.String getCorrelationID()
```

Returns the annotation ID used to calculate correlations.

Returns:
the annotation ID

getDataMax

```
public double getDataMax()
```

Returns the largest data value in this dataset.

Returns:
the maximum value

getDataMin

```
public double getDataMin()
```

Returns the smallest data value in this dataset.

Returns:
the minimum value

getSamplenames


```
public java.util.List getSamplenames()
```

Returns the samplenames.

Returns:
a List containing the samplenames.

getSize

```
public int getSize()
```

Returns the number of data values in this dataset.

Returns:
size

isEmpty

```
public boolean isEmpty()
```

Returns true if this dataset contains no values.

Returns:
true if this dataset contains no values.

main

```
public static void main(java.lang.String[] args)  
    throws java.lang.Exception
```

This method allows for testing the class on the command line. arguments to pass in:
datafilename, annotfilename, annotType, logValues, maxPercentile, dupmethod, correlationID

Throws:
java.lang.Exception

readDataFile

```
public void readDataFile(java.io.File datafile,  
                          java.io.File annotFile,  
                          java.lang.String annotType,  
                          boolean logValues,  
                          double maxPercentile,  
                          java.lang.String dupmethod,  
                          boolean calcRatios,  
                          double missingValueThreshold,  
                          double percAllowedMissing)  
    throws java.io.FileNotFoundException,  
           java.io.IOException
```

Reads in a datafile.

Parameters:

`datafile` - the File to import.
`annotFile` - the File containing gene annotations of the data
`annotType` - which annotation in the annotation file corresponds the gene annotation in the pathway
`logValues` - true if the datafile contains log values, false otherwise
`maxPercentile` - the data percentile at which the color is most intense
`dupmethod` - method used to resolve duplicates
`calcRatios` - if true, the values in the file need to have ratios calculated
`missingValueThreshold` - the minimum threshold for valid data values
`percAllowedMissing` - the percent of data values allowed to be missing for the gene to be represented

Throws:

`java.io.FileNotFoundException` - if the file cannot be found
`java.io.IOException` - if the file cannot be read

setCorrelationID

```
public boolean setCorrelationID(java.lang.String id)
```

Sets the annotation ID for which to calculate correlations.

Parameters:

`id` - The annotation ID

Returns:

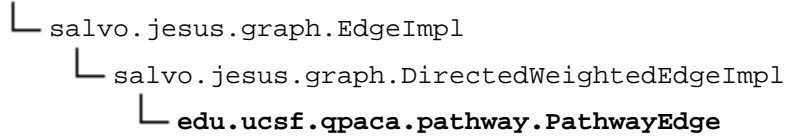
true if the ID exists in the dataset, false otherwise.



edu.ucsf.qpaca.pathway

Class PathwayEdge

java.lang.Object



All Implemented Interfaces:

[PathwayComponent](#), java.io.Serializable, salvo.jesus.graph.DirectedEdge, salvo.jesus.graph.DirectedWeightedEdge, salvo.jesus.graph.Edge, salvo.jesus.graph.GraphComponent, salvo.jesus.graph.LabeledEdge, salvo.jesus.graph.LabeledGraphComponent, salvo.jesus.graph.WeightedEdge

```
public class PathwayEdge
```

```
extends salvo.jesus.graph.DirectedWeightedEdgeImpl
implements PathwayComponent, java.io.Serializable
```

An edge in a pathway.

See Also:

[Serialized Form](#)

Field Summary	
static int	ACTIVATION_TYPE PathwayEdge type: activation.
protected java. util. HashMap	attributes attributes describing this edge
static int	INHIBITION_TYPE PathwayEdge type: inhibition.

static int	NEUTRAL_TYPE PathwayEdge type: neutral.
protected int	type PathwayEdges can be either positive (ACTIVATION_TYPE), negative (INHIBITION_TYPE) or neutral (NEUTRAL_TYPE).

Fields inherited from class salvo.jesus.graph.EdgeImpl	
str, vertexA, vertexB	

Fields inherited from interface salvo.jesus.graphDirectedEdge	
DIRECTION_A_TO_B, DIRECTION_B_TO_A, NODIRECTION	

Constructor Summary	
PathwayEdge (salvo.jesus.graph.Vertex sourceVertex, salvo.jesus.graph.Vertex sinkVertex) Creates a PathwayEdge object whose origin and destination vertices are specified by the method parameters.	
PathwayEdge (salvo.jesus.graph.Vertex sourceVertex, salvo.jesus.graph.Vertex sinkVertex, double weight) Creates a PathwayEdge object whose source and sink vertices and weight are specified by the parameters.	
PathwayEdge (salvo.jesus.graph.Vertex sourceVertex, salvo.jesus.graph.Vertex sinkVertex, int type) Creates a PathwayEdge object whose origin and destination vertices are specified by the method parameters.	

Method Summary	
protected java.lang.Object	clone () Creates a clone of this Edge.
java.lang.Object	getAttribute (java.lang.String type) Returns the attribute mapped to the specified attribute type.
java.util.Map	getAttributes () Returns the attributes of this Edge.
java.lang.String	getID () Returns the ID of this PathwayComponent.
int	getType () Returns the type of the edge.

boolean	hasAttribute (java.lang.String type) Tests to see if this PathwayComponent has this particular type of attribute.
java. lang. Object	removeAttribute (java.lang.String key) Removes a particular attribute.
void	setAttribute (java.lang.String key, java.lang.Object value) Creates a new attribute.
void	setType (int type) Sets the type of the edge.
java. lang. String	toString () Returns a String representation of the Edge, using the the toString() methods of Vertex.

Methods inherited from class salvo.jesus.graphDirectedWeightedEdgeImpl
getDirection, getSink, getSource, getWeight, setWeight

Methods inherited from class salvo.jesus.graphEdgeImpl
getLabel, getOppositeVertex, getVertexA, getVertexB, hasLabel, isFollowVertexLabel, setFollowVertexLabel, setLabel

Methods inherited from class java.lang.Object
equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface salvo.jesus.graphEdge
getOppositeVertex, getVertexA, getVertexB

Methods inherited from interface salvo.jesus.graphLabeledEdge
isFollowVertexLabel, setFollowVertexLabel

Methods inherited from interface salvo.jesus.graphLabeledGraphComponent
getLabel, hasLabel, setLabel

Methods inherited from interface salvo.jesus.graphEdge
getOppositeVertex, getVertexA, getVertexB

Methods inherited from interface salvo.jesus.graphLabeledEdge
isFollowVertexLabel, setFollowVertexLabel

Methods inherited from interface salvo.jesus.graph.LabeledGraphComponent

getLabel, hasLabel, setLabel

Field Detail

ACTIVATION_TYPE

```
public static final int ACTIVATION_TYPE
```

PathwayEdge type: activation.

See Also:

[Constant Field Values](#)

attributes

```
protected java.util.HashMap attributes
```

attributes describing this edge

INHIBITION_TYPE

```
public static final int INHIBITION_TYPE
```

PathwayEdge type: inhibition.

See Also:

[Constant Field Values](#)

NEUTRAL_TYPE

```
public static final int NEUTRAL_TYPE
```

PathwayEdge type: neutral.

See Also:

[Constant Field Values](#)

type

```
protected int type
```

PathwayEdges can be either positive (ACTIVATION_TYPE), negative (INHIBITION_TYPE) or neutral (NEUTRAL_TYPE).

Constructor Detail

PathwayEdge

```
public PathwayEdge(salvo.jesus.graph.Vertex sourceVertex,  
                    salvo.jesus.graph.Vertex sinkVertex)
```

Creates a PathwayEdge object whose origin and destination vertices are specified by the method parameters.

Parameters:

sourceVertex -
sinkVertex -

PathwayEdge

```
public PathwayEdge(salvo.jesus.graph.Vertex sourceVertex,  
                    salvo.jesus.graph.Vertex sinkVertex,  
                    double weight)
```

Creates a PathwayEdge object whose source and sink vertices and weight are specified by the parameters.

Parameters:


```
sourceVertex -  
sinkVertex -  
weight -
```

PathwayEdge

```
public PathwayEdge(salvo.jesus.graph.Vertex sourceVertex,  
                  salvo.jesus.graph.Vertex sinkVertex,  
                  int type)
```

Creates a PathwayEdge object whose origin and destination vertices are specified by the method parameters.

Parameters:

```
sourceVertex -  
sinkVertex -  
type -
```

Method Detail

clone

```
protected java.lang.Object clone()
```

Creates a clone of this Edge. This calls the Edge constructor, thereby creating a new instance of Edge. However, the vertices in both endpoints of the Edge are not cloned.

Overrides:

```
clone in class salvo.jesus.graph.EdgeImpl
```

Returns:

A clone of an instance of Edge.

getAttribute

```
public java.lang.Object getAttribute(java.lang.String type)
```

Returns the attribute mapped to the specified attribute type.

Specified by:

[getAttribute](#) in interface [PathwayComponent](#)

Parameters:

type - the attribute type

Returns:

the attribute specified by the type

getAttributes

```
public java.util.Map getAttributes()
```

Returns the attributes of this Edge.

Specified by:

[getAttributes](#) in interface [PathwayComponent](#)

Returns:

the attributes

getID

```
public java.lang.String getID()
```

Returns the ID of this PathwayComponent. In this case...return null since PathwayEdges do not have IDs.

Specified by:

[getID](#) in interface [PathwayComponent](#)

Returns:

null

getType

```
public int getType()
```

Returns the type of the edge.

Returns:

the type

hasAttribute

```
public boolean hasAttribute(java.lang.String type)
```

Tests to see if this PathwayComponent has this particular type of attribute.

Specified by:

[hasAttribute](#) in interface [PathwayComponent](#)

Parameters:

type - the attribute type

Returns:

true if this type of attribute exists;

removeAttribute

```
public java.lang.Object removeAttribute(java.lang.String key)
```

Removes a particular attribute.

Specified by:

[removeAttribute](#) in interface [PathwayComponent](#)

Parameters:

key - the attribute type

setAttribute

```
public void setAttribute(java.lang.String key,  
                           java.lang.Object value)
```

Creates a new attribute.

Specified by:

[setAttribute](#) in interface [PathwayComponent](#)

Parameters:

key - the attribute type

value - the attribute

setType

```
public void setType(int type)
```

Sets the type of the edge.

Parameters:

type - the new type

toString

```
public java.lang.String toString()
```

Returns a String representation of the Edge, using the the toString() methods of Vertex. Activation edges are represented as "-->"; inhibition edges are represented as "--|", and neutral edges are represented as "---".

Overrides:

toString in class salvo.jesus.graph.DirectWeightedEdgeImpl

Returns:

The String representation of the Edge

See Also:

Vertex

edu.ucsf.qpaca.pathway

Class PathwayElement

java.lang.Object

└─ edu.ucsf.qpaca.pathway.PathwayElement

All Implemented Interfaces:

[PathwayComponent](#), java.io.Serializable, java.lang.Comparable

```
public class PathwayElement
```

```
extends java.lang.Object
```

```
implements PathwayComponent, java.lang.Comparable, java.io.Serializable
```

An element of a pathway. PathwayElements are the smallest unit in a pathway and may represent both any subprocess of the pathway (such as another pathway) and, more importantly, any physical object that participates in the pathway.

See Also:

[Serialized Form](#)

Field Summary	
protected java. util. HashMap	attributes A hashtable of attributes.
protected java. lang. String	id the unique id
protected java. lang. String	type the type of this element.

Constructor Summary

PathwayElement () Constructs a new, empty element.
PathwayElement (java.lang.String newType, java.lang.String newID) Constructs a new Element object with the specified name and type.

Method Summary

int	compareTo (java.lang.Object o) Compares this Element to another Object.
java.lang.Object	getAttribute (java.lang.String type) Returns the attribute mapped to the specified attribute type.
java.util.Map	getAttributes () Returns the attributes of this Element.
java.lang.String	getID () Returns this element's value.
java.lang.String	getType () Returns this element's type.
boolean	hasAttribute (java.lang.String type) Tests to see if this PathwayComponent has this type of attribute.
java.lang.Object	removeAttribute (java.lang.String key) Removes a particular attribute
void	setAttribute (java.lang.String key, java.lang.Object value) Creates a new attribute
void	setType (java.lang.String newType) Sets this element's type.
java.lang.String	toString () Returns the name of this element.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

attributes

protected java.util.HashMap **attributes**

A hashtable of attributes.

id

protected java.lang.String **id**

the unique id

type

protected java.lang.String **type**

the type of this element. Such as "protein", "DNA", "small molecule", "process".

Constructor Detail

PathwayElement

public **PathwayElement**()

Constructs a new, empty element.

PathwayElement

public **PathwayElement**(java.lang.String newType,
java.lang.String newID)

Constructs a new Element object with the specified name and type.

Parameters:

newType - the type of this element

newID - the ID of this element

Method Detail

compareTo

```
public int compareTo(java.lang.Object o)
```

Compares this Element to another Object. If the Object is another Element, returns 0 if the two Elements have identical types and ids.

Specified by:

compareTo in interface `java.lang.Comparable`

Returns:

the value 0 if the argument is an equivalent Element

Throws:

`java.lang.ClassCastException` - if the argument is not a valid Element

getAttribute

```
public java.lang.Object getAttribute(java.lang.String type)
```

Returns the attribute mapped to the specified attribute type.

Specified by:

[getAttribute](#) in interface [PathwayComponent](#)

Parameters:

type - the attribute type

Returns:

the attribute specified by the type

getAttributes

```
public java.util.Map getAttributes()
```

Returns the attributes of this Element.

Specified by:

[getAttributes](#) in interface [PathwayComponent](#)

Returns:
the attributes

getID

```
public java.lang.String getID()
```

Returns this element's value.

Specified by:
[getID](#) in interface [PathwayComponent](#)

Returns:
the value

getType

```
public java.lang.String getType()
```

Returns this element's type.

Returns:
the type

hasAttribute

```
public boolean hasAttribute(java.lang.String type)
```

Tests to see if this PathwayComponent has this type of attribute.

Specified by:
[hasAttribute](#) in interface [PathwayComponent](#)

Parameters:
type - the attribute type

Returns:
true if this type of attribute exists;

removeAttribute

```
public java.lang.Object removeAttribute(java.lang.String key)
```

Removes a particular attribute

Specified by:

[removeAttribute](#) in interface [PathwayComponent](#)

Parameters:

key - the attribute type

setAttribute

```
public void setAttribute(java.lang.String key,  
                           java.lang.Object value)
```

Creates a new attribute

Specified by:

[setAttribute](#) in interface [PathwayComponent](#)

Parameters:

key - the attribute type
value - the attribute

setType

```
public void setType(java.lang.String newType)
```

Sets this element's type.

Parameters:

newType -

toString

```
public java.lang.String toString()
```

Returns the name of this element. If it does not have a specified name, returns the ID.

Overrides:

toString in class java.lang.Object

Returns:

the "name" attribute or, if no name exists, the ID attribute

edu.ucsf.qpaca.pathway

Class PathwayFactory

java.lang.Object

└─ edu.ucsf.qpaca.pathway.PathwayFactory

All Implemented Interfaces:

java.io.Serializable, salvo.jesus.graph.GraphFactory

```
public class PathwayFactory
```

```
extends java.lang.Object
```

```
implements salvo.jesus.graph.GraphFactory
```

The factory for creating Vertices and Edges in a Pathway class.

See Also:

[Serialized Form](#)

Constructor Summary

[PathwayFactory](#)()

Method Summary

salvo.jesus.graph.Edge	createEdge (salvo.jesus.graph.Vertex v1, salvo.jesus.graph.Vertex v2)
------------------------	---

salvo.jesus.graph.Vertex	createVertex ()
--------------------------	---------------------------------

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll,  
toString, wait, wait, wait
```

Constructor Detail

PathwayFactory

```
public PathwayFactory()
```

Method Detail

createEdge

```
public salvo.jesus.graph.Edge createEdge(salvo.jesus.graph.Vertex v1,  
                                           salvo.jesus.graph.Vertex v2)
```

Specified by:

createEdge in interface salvo.jesus.graph.GraphFactory

createVertex

```
public salvo.jesus.graph.Vertex createVertex()
```

Specified by:

createVertex in interface salvo.jesus.graph.GraphFactory

edu.ucsf.qpaca.pathway

Class PathwaySVG

java.lang.Object

└─ edu.ucsf.qpaca.pathway.PathwaySVG

```
public class PathwaySVG
```

```
extends java.lang.Object
```

Handles manipulation and coloring (by data value) of SVG files generated by GraphViz.

Constructor Summary

[PathwaySVG\(\)](#)

[PathwaySVG](#)(java.lang.String uri)

[PathwaySVG](#)(java.lang.String uri, java.io.File datafile, java.io.File annotFile, java.lang.String annotType, boolean logValues, double percentile, java.lang.String dupmethod, boolean calcRatios, double missingValueThreshold, double percAllowedMissing)
Creates a new PathwaySVG object with the specified SVG file and dataset.

Method Summary

protected double	calculateColor (double value)
protected double	calculateColor (double value, double dMax, double dMin, double dMid)
org.w3c. dom.svg. SVGDocument	color (java.util.Map dataForColoring, int index, java.lang.String titleString, double dMax, double dMin, double dMid, java.lang.String corrID) Colors the svg document based on values in at a specific index in the data list
org.w3c. dom.svg. SVGDocument	colorByCorrelation ()
org.w3c. dom.svg. SVGDocument	colorBySample (int sampleIndex)
boolean	generateCorrelationData ()

java.lang. String	getCorrelationID()
double	getDataMax()
double	getDataMin()
org.w3c. dom.svg. SVGDocument	getDocument()
boolean	isTweaked()
static void	main (java.lang.String[] args) This method allows for testing the class on the command line.
void	readDataFile (java.io.File datafile, java.io.File annotFile, java.lang.String annotType, boolean logValues, double maxPercentile, java.lang.String dupmethod, boolean calcRatios, double missingValueThreshold, double percAllowedMissing) Reads in a datafile that will be used in generating colored images.
boolean	setCorrelationID (java.lang.String id)
void	setDocument (java.lang.String uri)
void	tweak (java.lang.String hreftag) Takes the svg file output by dot and separates the rectangles.
static void	writeSVGFile (org.w3c.dom.svg.SVGDocument doc, java.lang.String filename) Writes out the SVG document.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PathwaySVG

```
public PathwaySVG()
```

PathwaySVG

```
public PathwaySVG(java.lang.String uri)
    throws java.io.IOException
```

Parameters:

uri - the svg document URI

Throws:

java.io.IOException

PathwaySVG

```
public PathwaySVG(java.lang.String uri,
                  java.io.File datafile,
                  java.io.File annotFile,
                  java.lang.String annotType,
                  boolean logValues,
                  double percentile,
                  java.lang.String dupmethod,
                  boolean calcRatios,
                  double missingValueThreshold,
                  double percAllowedMissing)
    throws java.io.FileNotFoundException,
           java.io.IOException
```

Creates a new PathwaySVG object with the specified SVG file and dataset. The data is contained a datafile (with genes in rows and samples in columns). Gene identifiers must be located in the first column. Missing data in the file is identified using the empty string or "NA". Gene annotations are in a corresponding annotation file, which also contains identifiers in the first column. The annotation type and list of samples further describe the data set.

Parameters:

uri - the svg document URI
datafile - the data file
annotFile - the annotation file
annotType - the annotation type that corresponds to the annotation type in the svg document hrefs.
logValues - true if the datafile contains log values, false otherwise
percentile - the percentile of the max/min threshold at which color is most intense
dupmethod - method used to resolve duplicates
calcRatios - if true, the values in the file need to have ratios calculated
missingValueThreshold - the minimum threshold for valid data values
percAllowedMissing - the percent of data values allowed to be missing for the gene to be represented

Throws:

FileNotFoundException
java.io.IOException

Method Detail

calculateColor

```
protected double calculateColor(double value)
```

calculateColor

```
protected double calculateColor(double value,
                                double dMax,
                                double dMin,
```


double dMid)

color

```
public org.w3c.dom.svg.SVGDocument color(java.util.Map dataForColoring,  
                                           int index,  
                                           java.lang.String titleString,  
                                           double dMax,  
                                           double dMin,  
                                           double dMid,  
                                           java.lang.String corrID)
```

Colors the svg document based on values in at a specific index in the data list

Parameters:

dataForColoring - dataset
index - index of sample color
titleString - title of the svg plot
dMax - data maximum
dMin - data minimum
dMid - data midpoint
corrID - ID of correlation gene

colorByCorrelation

```
public org.w3c.dom.svg.SVGDocument colorByCorrelation()
```

colorBySample

```
public org.w3c.dom.svg.SVGDocument colorBySample(int sampleIndex)
```

generateCorrelationData

```
public boolean generateCorrelationData()
```

getCorrelationID

```
public java.lang.String getCorrelationID()
```

getDataMax

```
public double getDataMax()
```

getDataMin

```
public double getDataMin()
```

getDocument

```
public org.w3c.dom.svg.SVGDocument getDocument()
```

isTweaked

```
public boolean isTweaked()
```

main

```
public static void main(java.lang.String[] args)  
    throws java.lang.Exception
```

This method allows for testing the class on the command line. args to pass in: svgfilename, datafilename, annotfilename,

Throws:
java.lang.Exception

readDataFile

```
public void readDataFile(java.io.File datafile,  
    java.io.File annotFile,  
    java.lang.String annotType,  
    boolean logValues,  
    double maxPercentile,  
    java.lang.String dupmethod,  
    boolean calcRatios,  
    double missingValueThreshold,  
    double percAllowedMissing)
```

throws `java.io.FileNotFoundException`,
`java.io.IOException`

Reads in a datafile that will be used in generating colored images.

Parameters:

`datafile` - the File to import.
`annotFile` - the File containing gene annotations of the data
`annotType` - which annotation in the annotation file corresponds the gene annotation in the SVG file
`logValues` - true if the datafile contains log values, false otherwise
`maxPercentile` - the data percentile at which the color is most intense
`dupmethod` - method used to resolve duplicates
`calcRatios` - if true, the values in the file need to have ratios calculated
`missingValueThreshold` - the minimum threshold for valid data values
`percAllowedMissing` - the percent of data values allowed to be missing for the gene to be represented

Throws:

`java.io.FileNotFoundException` - if the file cannot be found
`java.io.IOException` - if the file cannot be read

setCorrelationID

```
public boolean setCorrelationID(java.lang.String id)
```

setDocument

```
public void setDocument(java.lang.String uri)  
    throws java.io.IOException
```

Throws:

`java.io.IOException`

tweak

```
public void tweak(java.lang.String hreftag)
```

Takes the svg file output by dot and separates the rectangles.

Parameters:

`hreftag` - The portion of the href tag that starts the gene id section.

writeSVGFile

```
public static void writeSVGFile(org.w3c.dom.svg.SVGDocument doc,
```

```
        java.lang.String filename)  
throws javax.xml.transform.TransformerConfigurationException,  
        javax.xml.transform.TransformerException
```

Writes out the SVG document.

Parameters:

filename - the name of the output file

Throws:

javax.xml.transform.TransformerConfigurationException
javax.xml.transform.TransformerException

edu.ucsf.qpaca.pathway
Class PathwayUtil

java.lang.Object
└─ edu.ucsf.qpaca.pathway.PathwayUtil

public class **PathwayUtil**

extends java.lang.Object

Static utility methods for manipulating data values.

Constructor Summary

[PathwayUtil\(\)](#)

Method Summary

static java. lang.Object	arrayExpand (java.lang.Object a, int newLength) Resizes an array.
static java. util.List	indexOfNA (double[] values) Determines the index of all "NaN" values in an array of doubles.
static java. lang.String	joinDoubleArray (double[] tojoin, java.lang.String token) Join the values in an array of values into a string with the specified separator token.
static double	mean (double[] values) Determines the mean of the values in a double array.
static double	pearsons (double[] data1, double[] data2, int n) Calculates the pearson's correlation coefficient between two arrays data values of the length: data1 and data2 derived from "Numerical Recipes in C", pg 638-639
static double	randomizedSelect (double[] array, int begin, int end, int k) Selects the kth largest element from an array.

static double []	removeIndices (double[] values, java.util.Set indices) Removes the specified set of indices from an array of doubles.
static double []	removeNA (double[] values) removes the NaN values from an array of doubles.
static java. util.List	removeNA (java.util.List values) removes the NaN values from an List
static java. lang.String[]	removeNA (java.lang.String[] values) removes the null values from an array of Strings.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PathwayUtil

```
public PathwayUtil()
```

Method Detail

arrayExpand

```
public static java.lang.Object arrayExpand(java.lang.Object a,
                                           int newLength)
```

Resizes an array.

Parameters:

a - the array to resize

newLength - the length of the new array (if newLength < a.length, newLength is set to length+length/2).

Returns:

the new array or null if a was not an array.

indexOfNA

```
public static java.util.List indexOfNA(double[] values)
```

Determines the index of all "NaN" values in an array of doubles.

Parameters:

values - the array

Returns:

a List of indices in the array

joinDoubleArray

```
public static java.lang.String joinDoubleArray(double[] tojoin,  
                                                java.lang.String token)
```

Join the values in an array of values into a string with the specified separator token.

Parameters:

tojoin - the array

token - the string separator

Returns:

the resulting string representation of the array

mean

```
public static double mean(double[] values)
```

Determines the mean of the values in a double array. If there are NaN values in the array, they are removed first and the mean is calculated without them.

Parameters:

values - an array of doubles

Returns:

the mean of the values

pearsons

```
public static double pearsons(double[] data1,
```

```
double[] data2,  
int n)
```

Calculates the pearson's correlation coefficient between two arrays data values of the length: data1 and data2 derived from "Numerical Recipes in C", pg 638-639

Parameters:

data1 - an array
data2 - a second array
n - the length of the two arrays

Returns:

the pearson's correlation coefficient

randomizedSelect

```
public static double randomizedSelect(double[] array,  
int begin,  
int end,  
int k)  
throws java.lang.IllegalArgumentException
```

Selects the kth largest element from an array. To calculate the median, set $k=(end-begin)/2$

Parameters:

array - the array to search
begin - index of the first element in the array
end - index of the last element in the array
k - the index to select

Throws:

java.lang.IllegalArgumentException

removeIndices

```
public static double[] removeIndices(double[] values,  
java.util.Set indices)
```

Removes the specified set of indices from an array of doubles.

Parameters:

values - the array
indices - the indicies in the array to remove

Returns:

the modified array

removeNA

```
public static double[] removeNA(double[] values)
```

removes the NaN values from an array of doubles.

Parameters:

values - the array to process

Returns:

the new double array

removeNA

```
public static java.util.List removeNA(java.util.List values)
```

removes the NaN values from an List

Parameters:

values - the List to process

Returns:

the new List

removeNA

```
public static java.lang.String[] removeNA(java.lang.String[] values)
```

removes the null values from an array of Strings.

Parameters:

values - the array to process

Returns:

the new String array

edu.ucsf.qpaca.pathway
Class PathwayVertex

```
java.lang.Object
├── salvo.jesus.graph.VertexImpl
│   └── edu.ucsf.qpaca.pathway.PathwayVertex
```

All Implemented Interfaces:

[PathwayComponent](#), java.io.Serializable, java.lang.Comparable, salvo.jesus.graph.GraphComponent, salvo.jesus.graph.LabeledGraphComponent, salvo.jesus.graph.Vertex

```
public class PathwayVertex
```

```
extends salvo.jesus.graph.VertexImpl
implements PathwayComponent, java.lang.Comparable, java.io.Serializable
```

A vertex in a pathway. This class encapsulates an object that the vertex will represent. Hence, a PathwayVertex can represent any object that extends java.lang.Object by simply calling setObject() or specifying the object in the constructor.

See Also:

[Serialized Form](#)

Field Summary	
protected java.util. HashMap	attributes

Fields inherited from class salvo.jesus.graph.VertexImpl
object

Constructor Summary

PathwayVertex ()	Creates an empty PathwayVertex
PathwayVertex (java.lang.Object newobject)	Creates a new PathwayVertex that contains a specified object.
PathwayVertex (java.lang.Object newobject, java.lang.String newType)	Creates a new PathwayVertex object that contains a specific object with a specific type.
PathwayVertex (java.lang.Object newobject, java.lang.String newType, java.util.Map newAttributes)	Creates a new PathwayVertex object that contains a specific object with a specific type and attributes.
PathwayVertex (java.lang.Object newobject, java.lang.String newType, java.util.Map newAttributes, java.lang.String newID)	Creates a new PathwayVertex object that contains a specific object with a specific type, attributes, and id.
PathwayVertex (java.lang.Object newobject, java.lang.String newType, java.lang.String newID)	Creates a new PathwayVertex object that contains a specific object with a specific type and id.
PathwayVertex (java.lang.String newType)	Creates an empty PathwayVertex with the specified type.
PathwayVertex (java.lang.String newType, java.lang.String newID)	Creates an empty PathwayVertex with the specified type and id.

Method Summary	
int	compareTo (java.lang.Object o) Compares this PathwayVertex to another Object.
java.lang.Object	getAttribute (java.lang.String type) Returns the attribute mapped to the specified attribute type.
java.util.Map	getAttributes () Returns the attributes of this Vertex.
java.util.Set	getElements () Retrieves a List of all PathwayElements within this Vertex
java.lang.String	getID () Returns the ID of this PathwayComponent.
java.lang.String	getType () Returns the type of this PathwayVertex.
boolean	hasAttribute (java.lang.String type) Tests to see if this PathwayComponent has this type of attribute.

boolean	hasObject() Tests to see if this PathwayVertex contains an object.
boolean	hasType() Tests to see if this PathwayVertex has a type.
java. lang. Object	removeAttribute() (java.lang.String key) Removes the specified attribute.
void	setAttribute() (java.lang.String key, java.lang.Object value) Creates a new attribute.
java. lang. String	toString() Creates a string representation of this PathwayVertex: id:type:object

Methods inherited from class salvo.jesus.graph.VertexImpl

getLabel, getObject, hasLabel, setLabel, setObject

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

attributes

protected java.util.HashMap **attributes**

Constructor Detail

PathwayVertex

public **PathwayVertex()**

Creates an empty PathwayVertex

PathwayVertex

```
public PathwayVertex(java.lang.Object newobject)
```

Creates a new PathwayVertex that contains a specified object.

Parameters:

newobject -

PathwayVertex

```
public PathwayVertex(java.lang.Object newobject,  
                      java.lang.String newType)
```

Creates a new PathwayVertex object that contains a specific object with a specific type.

Parameters:

newobject - the object that the PathwayVertex will encapsulate
newType - the new type

PathwayVertex

```
public PathwayVertex(java.lang.Object newobject,  
                      java.lang.String newType,  
                      java.util.Map newAttributes)
```

Creates a new PathwayVertex object that contains a specific object with a specific type and attributes.

Parameters:

newobject - the object that the PathwayVertex will encapsulate
newType - the new type

PathwayVertex

```
public PathwayVertex(java.lang.Object newobject,  
                      java.lang.String newType,  
                      java.util.Map newAttributes,  
                      java.lang.String newID)
```

Creates a new PathwayVertex object that contains a specific object with a specific type, attributes, and id.

Parameters:

newobject - the object that the PathwayVertex will encapsulate
newType - the new type

PathwayVertex

```
public PathwayVertex(java.lang.Object newobject,  
                     java.lang.String newType,  
                     java.lang.String newID)
```

Creates a new PathwayVertex object that contains a specific object with a specific type and id.

Parameters:

newobject - the object that the PathwayVertex will encapsulate
newType - the new type

PathwayVertex

```
public PathwayVertex(java.lang.String newType)
```

Creates an empty PathwayVertex with the specified type.

Parameters:

newType - this vertex's type

PathwayVertex

```
public PathwayVertex(java.lang.String newType,  
                     java.lang.String newID)
```

Creates an empty PathwayVertex with the specified type and id.

Parameters:

newType - this vertex's type
newID - this vertex's id

Method Detail

compareTo

```
public int compareTo(java.lang.Object o)
```

Compares this PathwayVertex to another Object. If the Object is another PathwayVertex, returns 0 if the corresponding Assemblies and types are equivalent.

Specified by:

compareTo in interface `java.lang.Comparable`

Parameters:

o - the object to be tested.

Returns:

0 if the argument is an equivalent PathwayVertex, 1 otherwise

Throws:

`java.lang.ClassCastException` - if the argument is not a valid PathwayVertex

getAttribute

```
public java.lang.Object getAttribute(java.lang.String type)
```

Returns the attribute mapped to the specified attribute type.

Specified by:

[getAttribute](#) in interface [PathwayComponent](#)

Parameters:

type - the attribute type

Returns:

the attribute specified by the type

getAttributes

```
public java.util.Map getAttributes()
```

Returns the attributes of this Vertex.

Specified by:

[getAttributes](#) in interface [PathwayComponent](#)

Returns:

the attributes

getElements

```
public java.util.Set getElements()
```

Retrieves a List of all PathwayElements within this Vertex

Returns:

a List of PathwayElements

getID

```
public java.lang.String getID()
```

Returns the ID of this PathwayComponent.

Specified by:

[getID](#) in interface [PathwayComponent](#)

Returns:

the ID

getType

```
public java.lang.String getType()
```

Returns the type of this PathwayVertex.

Returns:

a String representing the type.

hasAttribute

```
public boolean hasAttribute(java.lang.String type)
```

Tests to see if this PathwayComponent has this type of attribute.

Specified by:

[hasAttribute](#) in interface [PathwayComponent](#)

Parameters:

type - the attribute type

Returns:

true if this type of attribute exists;

hasObject

```
public boolean hasObject()
```

Tests to see if this PathwayVertex contains an object.

Returns:

true if the object exists;

hasType

```
public boolean hasType()
```

Tests to see if this PathwayVertex has a type.

Returns:

true if it has a type.

removeAttribute

```
public java.lang.Object removeAttribute(java.lang.String key)
```

Removes the specified attribute.

Specified by:

[removeAttribute](#) in interface [PathwayComponent](#)

Parameters:

key - the attribute type

setAttribute

```
public void setAttribute(java.lang.String key,  
                           java.lang.Object value)
```

Creates a new attribute.

Specified by:

[setAttribute](#) in interface [PathwayComponent](#)

Parameters:

key - the attribute type
value - the attribute

toString

```
public java.lang.String toString()
```

Creates a string representation of this PathwayVertex: id:type:object

Overrides:

toString in class salvo.jesus.graph.VertexImpl

Returns:

String representation

edu.ucsf.qpaca.pathway.algorithm

Class PathwayMinimumSpanningTree

java.lang.Object

└─ edu.ucsf.qpaca.pathway.algorithm.PathwayMinimumSpanningTree

All Implemented Interfaces:

java.io.Serializable

```
public class PathwayMinimumSpanningTree
```

```
extends java.lang.Object
```

```
implements java.io.Serializable
```

A concrete implementation of the minimum spanning tree algorithm using Kruskal's method specifically geared to Pathway objects.

See Also:

[Serialized Form](#)

Constructor Summary

[PathwayMinimumSpanningTree](#)([Pathway](#) path)

Creates an instance of PathwayMinimumSpanningTree

Method Summary

salvo.jesus. graph. WeightedGraph	minimumSpanningTree () Determine the minimum spanning tree of a pathway using Kruskal's method.
---	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PathwayMinimumSpanningTree

```
public PathwayMinimumSpanningTree(Pathway path)
```

Creates an instance of PathwayMinimumSpanningTree

Parameters:

path - The Pathway where the minimum spanning tree will be determined.

Method Detail

minimumSpanningTree

```
public salvo.jesus.graph.WeightedGraph minimumSpanningTree()
```

Determine the minimum spanning tree of a pathway using Kruskal's method.

edu.ucsf.qpaca.pathway.algorithm

Class ShortestPathAlgorithm

java.lang.Object

└─ edu.ucsf.qpaca.pathway.algorithm.ShortestPathAlgorithm

All Implemented Interfaces:

java.io.Serializable

Direct Known Subclasses:

[ShortestPathDijkstraAlgorithm](#)

```
public abstract class ShortestPathAlgorithm
```

```
extends java.lang.Object
```

```
implements java.io.Serializable
```

Abstract class for implementing the shortest path algorithm. Concrete subclasses must never modify the graph when it is computing the shortestpath.

See Also:

[Serialized Form](#)

Constructor Summary

[ShortestPathAlgorithm](#)([Pathway](#) path)

Method Summary

abstract salvo.jesus. graph. WeightedGraph	shortestPath (salvo.jesus.graph.Vertex from) Abstract method to be implemented by subclasses to determine a shortest path spanning tree from a given vertex in the form of a graph.
---	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ShortestPathAlgorithm

```
public ShortestPathAlgorithm(Pathway path)
```

Method Detail

shortestPath

```
public abstract salvo.jesus.graph.WeightedGraph shortestPath(salvo.jesus.graph.Vertex from)
```

Abstract method to be implemented by subclasses to determine a shortest path spanning tree from a given vertex in the form of a graph.

Returns:

A new Pathway that represents the shortest path spanning tree of the original Pathway. **Do not** modify the contents of the returned Pathway.

edu.ucsf.qpaca.pathway.algorithm

Class ShortestPathDijkstraAlgorithm

java.lang.Object

└─ [edu.ucsf.qpaca.pathway.algorithm.ShortestPathAlgorithm](#)

└─ [edu.ucsf.qpaca.pathway.algorithm.ShortestPathDijkstraAlgorithm](#)

All Implemented Interfaces:

java.io.Serializable

```
public class ShortestPathDijkstraAlgorithm
```

```
extends ShortestPathAlgorithm
```

A concrete implementation of ShortestPathAlgorithm using Dijkstra's method.

See Also:

[Serialized Form](#)

Constructor Summary

[ShortestPathDijkstraAlgorithm](#)([Pathway](#) path, salvo.jesus.util.
HeapNodeComparator comparator)
Creates an instance of ShortestPathDijkstraAlgorithm.

Method Summary

salvo.jesus. graph. WeightedGraph	shortestPath (salvo.jesus.graph.Vertex from) Determines the shortest path from a given vertex to all other vertices that are in the same connected set as the given vertex in the weighted graph using Dijkstra's algorithm.
---	---

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ShortestPathDijkstraAlgorithm

```
public ShortestPathDijkstraAlgorithm(Pathway path,  
                                     salvo.jesus.util.HeapNodeComparator comparator)
```

Creates an instance of ShortestPathDijkstraAlgorithm.

Parameters:

`path` - The Pathway where a shortest path spanning tree will be determined.

`comparator` - The HeapNodeComparator to be used to compare priorities of objects in the fringe/heap.

Method Detail

shortestPath

```
public salvo.jesus.graph.WeightedGraph shortestPath(salvo.jesus.graph.Vertex from)
```

Determines the shortest path from a given vertex to all other vertices that are in the same connected set as the given vertex in the weighted graph using Dijkstra's algorithm.

Specified by:

[shortestPath](#) in class [ShortestPathAlgorithm](#)

Parameters:

`from` - The Vertex from where we want to obtain the shortest path to all other vertices.

Returns:

A Pathway comprising of the shortest path spanning tree.

edu.ucsf.qpaca.parser

Class MapBioPAXtoQPACA

java.lang.Object

└─ edu.ucsf.qpaca.parser.MapBioPAXtoQPACA

public class **MapBioPAXtoQPACA**

extends java.lang.Object

Maps between BioPAX keywords and QPACA keywords.

Field Summary	
static java. lang.String	<u>BIOPAX_EDGE_TYPE_ATTRIBUTE</u> Pathway Attribute: BioPAX Edge Type.
static java. lang.String	<u>BIOPAX_ID_ATTRIBUTE</u> Pathway Attribute: BioPAX ID.
static java. lang.String	<u>BIOPAX_NAME_ATTRIBUTE</u> Pathway Attribute: BioPAX Name.
static java. lang.String	<u>BIOPAX_NODE_TYPE_ATTRIBUTE</u> Pathway Attribute: BioPAX Node Type.
static java. lang.String	<u>COFACTOR</u> Pathway Edge Attribute: COFACTOR
static java. lang.String	<u>CONTAINS</u> Pathway Edge Attribute: CONTAINS
static java. lang.String	<u>CONTROLLED</u> Pathway Edge Attribute: CONTROLLED
static java. lang.String	<u>CONTROLLER</u> Pathway Edge Attribute: CONTROLLER
static java. lang.String	<u>LEFT</u> Pathway Edge Attribute: LEFT

static java. lang.String	PARTICIPANT Pathway Edge Attribute: PARTICIPANT
static java. lang.String	RIGHT Pathway Edge Attribute: RIGHT

Constructor Summary

[MapBioPAXtoQPACA](#)([BioPaxUtil](#) bpUtil, [Pathway](#) path)
Initializes the mapping.

Method Summary

void	doMapping () Executes the Mapping.
java. util. ArrayList	getWarningList () Retrieves a list of all exceptions and warnings encountered during execution.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

BIOPAX_EDGE_TYPE_ATTRIBUTE

```
public static final java.lang.String BIOPAX_EDGE_TYPE_ATTRIBUTE
```

Pathway Attribute: BioPAX Edge Type.

See Also:

[Constant Field Values](#)

BIOPAX_ID_ATTRIBUTE

```
public static final java.lang.String BIOPAX_ID_ATTRIBUTE
```

Pathway Attribute: BioPAX ID.

See Also:

[Constant Field Values](#)

BIOPAX_NAME_ATTRIBUTE

```
public static final java.lang.String BIOPAX_NAME_ATTRIBUTE
```

Pathway Attribute: BioPAX Name.

See Also:

[Constant Field Values](#)

BIOPAX_NODE_TYPE_ATTRIBUTE

```
public static final java.lang.String BIOPAX_NODE_TYPE_ATTRIBUTE
```

Pathway Attribute: BioPAX Node Type.

See Also:

[Constant Field Values](#)

COFACTOR

```
public static final java.lang.String COFACTOR
```

Pathway Edge Attribute: COFACTOR

See Also:

[Constant Field Values](#)

CONTAINS

public static final java.lang.String **CONTAINS**

Pathway Edge Attribute: CONTAINS

See Also:

[Constant Field Values](#)

CONTROLLED

public static final java.lang.String **CONTROLLED**

Pathway Edge Attribute: CONTROLLED

See Also:

[Constant Field Values](#)

CONTROLLER

public static final java.lang.String **CONTROLLER**

Pathway Edge Attribute: CONTROLLER

See Also:

[Constant Field Values](#)

LEFT

public static final java.lang.String **LEFT**

Pathway Edge Attribute: LEFT

See Also:

[Constant Field Values](#)

PARTICIPANT

```
public static final java.lang.String PARTICIPANT
```

Pathway Edge Attribute: PARTICIPANT

See Also:

[Constant Field Values](#)

RIGHT

```
public static final java.lang.String RIGHT
```

Pathway Edge Attribute: RIGHT

See Also:

[Constant Field Values](#)

Constructor Detail

MapBioPAXtoQPACA

```
public MapBioPAXtoQPACA(BioPaxUtil bpUtil,  
                        Pathway path)
```

Initializes the mapping.

Parameters:

bpUtil - BioPAX Utility Class
path - Pathway Object

Method Detail

doMapping

```
public void doMapping()
```

throws org.jdom.JDOMException,
java.lang.Exception

Executes the Mapping.

Throws:

org.jdom.JDOMException - Error Parsing XML via JDOM.
java.lang.Exception

getWarningList

public java.util.ArrayList **getWarningList**()

Retrieves a list of all exceptions and warnings encountered during execution.

Returns:

ArrayList of String Objects.

edu.ucsf.qpaca.parser

Class MapKGMLtoQPACA

java.lang.Object

└─ edu.ucsf.qpaca.parser.MapKGMLtoQPACA

```
public class MapKGMLtoQPACA
```

```
extends java.lang.Object
```

Maps between KEGG KGML keywords and QPACA keywords.

Constructor Summary

[MapKGMLtoQPACA](#)([KGMLUtil](#) kgmlUtil, [Pathway](#) path)

Initializes the mapping.

Method Summary

void	doMapping () Executes the Mapping.
java.util. ArrayList	getWarningList () Retrieves a list of all exceptions and warnings encountered during execution.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

MapKGMLtoQPACA

```
public MapKGMLtoQPACA(KGMLUtil kgmlUtil,  
                     Pathway path)
```

Initializes the mapping.

Parameters:

kgmlUtil - KGML Utility Class.
path - Pathway Object.

Method Detail

doMapping

```
public void doMapping()  
    throws org.jdom.JDOMException,  
           java.lang.Exception
```

Executes the Mapping.

Throws:

org.jdom.JDOMException - Error Parsing XML via JDOM.
java.lang.Exception

getWarningList

```
public java.util.ArrayList getWarningList()
```

Retrieves a list of all exceptions and warnings encountered during execution.

Returns:

ArrayList of String Objects.

edu.ucsf.qpaca.parser

Class ParseQPACALanguage

java.lang.Object

└─ edu.ucsf.qpaca.parser.ParseQPACALanguage

```
public class ParseQPACALanguage
```

```
extends java.lang.Object
```

Creates QPACA pathways from files written in the QPACA Pathway language.

Constructor Summary

ParseQPACALanguage (java.io.Reader reader)
Initializes the parser.

Method Summary

Pathway	getPathway () Retrieves the Pathway.
java. lang. String	getPathwayID () Retrieves the pathway name.
java. lang. String	getPathwayLabel () Retrieves the pathway label.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ParseQPACALanguage

```
public ParseQPACALanguage(java.io.Reader reader)
    throws java.io.IOException,
           java.lang.Exception
```

Initializes the parser.

Parameters:

reader - Reader Object.

Throws:

java.io.IOException - Input/Output Error.
java.lang.Exception

Method Detail

getPathway

```
public Pathway getPathway()
```

Retrieves the Pathway.

Returns:

Pathway

getPathwayID

```
public java.lang.String getPathwayID()
```

Retrieves the pathway name.

Returns:

pathway name

getPathwayLabel

```
public java.lang.String getPathwayLabel()
```

Retrieves the pathway label.

Returns:
pathway label



edu.ucsf.qpaca.parser
Class PathwayToDot

java.lang.Object
└─ edu.ucsf.qpaca.parser.PathwayToDot

public class **PathwayToDot**

extends java.lang.Object

Translates Pathways into GraphViz .dot files.

Constructor Summary

[PathwayToDot\(\)](#)

Default Constructor

Method Summary

void	createDotFile (Pathway path, java.lang.String fileName, java.lang.String linkID, java.lang.String linkBase) Creates a GraphViz .dot file.
protected java. util.List	writeAssemblies (java.util.Set assemblies, java.io. FileWriter out, java.lang.String linkID, java.lang. String linkBase) Writes out QPACA PathwayAssemblies, delegating the task to writeElement or writeRecord as necessary.
protected void	writeAttributes (java.util.Map atts, java.io.FileWriter out) Writes out label or name attributes.
protected void	writeHeader (java.io.FileWriter out) Writes out the header of the GraphViz file.

protected java. util.List	<p>writeNodes(java.util.Set vertices, Pathway path, java.io. FileWriter out, java.lang.String linkID, java.lang. String linkBase)</p> <p>Writes out QPACA PathwayVertices, specifically handles the those of the QPACAConstants.EVENT type, delegating the task of writing out the other Vertices to writeAssemblies after extracting lists of PathwayAssemblies.</p>
protected void	<p>writeRecord(PathwayAssembly a, java.io.FileWriter out, java. lang.String linkID, java.lang.String linkBase)</p> <p>Writes out a GraphViz Record, which represents a collection of PathwayElements found at the same node (stored as a PathwayAssembly of type QPACAConstants.ALIAS or QPACAConstants.FAMILY).</p>

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PathwayToDot

```
public PathwayToDot()
```

Default Constructor

Method Detail

createDotFile

```
public void createDotFile(Pathway path,  
                          java.lang.String fileName,  
                          java.lang.String linkID,  
                          java.lang.String linkBase)
```

Creates a GraphViz .dot file.

Parameters:

path - the pathway to output
 fileName - the output file
 linkID - attribute from which to create link
 linkBase - base of href link string

writeAssemblies

```
protected java.util.List writeAssemblies(java.util.Set assemblies,  
                                           java.io.FileWriter out,  
                                           java.lang.String linkID,  
                                           java.lang.String linkBase)  
    throws java.io.IOException
```

Writes out QPACA PathwayAssemblies, delegating the task to writeElement or writeRecord as necessary.

Parameters:

assemblies - all of the PathwayAssemblies
out - the output FileWriter
linkID - id of the element in the database link
linkBase - the base URL of the database link

Throws:

java.io.IOException - if the file cannot be accessed.

writeAttributes

```
protected void writeAttributes(java.util.Map atts,  
                                java.io.FileWriter out)  
    throws java.io.IOException
```

Writes out label or name attributes. Attributes for all PathwayComponents are handled the same way.

Parameters:

out - the output FileWriter
atts - the attributes

Throws:

java.io.IOException - if the file cannot be accessed.

writeHeader

```
protected void writeHeader(java.io.FileWriter out)  
    throws java.io.IOException
```

Writes out the header of the GraphViz file.

Parameters:

out - the output FileWriter

Throws:

java.io.IOException - if the file cannot be accessed.

writeNodes

```
protected java.util.List writeNodes(java.util.Set vertices,
                                   Pathway path,
                                   java.io.FileWriter out,
                                   java.lang.String linkID,
                                   java.lang.String linkBase)
    throws java.io.IOException
```

Writes out QPACA PathwayVertices, specifically handles the those of the QPACAConstants. EVENT type, delegating the task of writing out the other Vertices to writeAssemblies after extracting lists of PathwayAssemblies.

Parameters:

vertices - all of the PathwayVertices

path - the pathway

out - the output FileWriter

linkID - id of the element in the database link

linkBase - the base URL of the database link

Throws:

java.io.IOException - if the file cannot be accessed.

writeRecord

```
protected void writeRecord(PathwayAssembly a,
                           java.io.FileWriter out,
                           java.lang.String linkID,
                           java.lang.String linkBase)
    throws java.io.IOException
```

Writes out a GraphViz Record, which represents a collection of PathwayElements found at the same node (stored as a PathwayAssembly of type QPACAConstants.ALIAS or QPACAConstants.FAMILY).

Parameters:

a - the PathwayAssembly
out - the output FileWriter
linkID - id of the element in the database link
linkBase - the base URL of the database link

Throws:

`java.io.IOException` - if the file cannot be accessed.

edu.ucsf.qpaca.parser
Class TestBioPaxUtil

```
java.lang.Object
├── junit.framework.Assert
│   └── junit.framework.TestCase
│       └── edu.ucsf.qpaca.parser.TestBioPaxUtil
```

All Implemented Interfaces:
junit.framework.Test

```
public class TestBioPaxUtil
```

```
extends junit.framework.TestCase
```

Tests the BioPaxUtil Class.

Constructor Summary

[TestBioPaxUtil\(\)](#)

Method Summary

void	testUtil() Tests the BioPAX Utility Class using a sample biopax file located at sampleData/ biopax_sample1.owl
------	--

Methods inherited from class junit.framework.TestCase

countTestCases, createResult, getName, run, run, runBare, runTest, setName, setUp, tearDown, toString

Methods inherited from class junit.framework.Assert

```
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertFalse, assertFalse, assertNotNull, assertNotNull,
assertNotSame, assertNotSame, assertNull, assertNull, assertEquals,
assertSame, assertTrue, assertTrue, fail, fail
```

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,
wait, wait
```

Constructor Detail

TestBioPaxUtil

```
public TestBioPaxUtil()
```

Method Detail

testUtil

```
public void testUtil()
    throws java.lang.Exception
```

Tests the BioPAX Utility Class using a sample biopax file located at sampleData/
biopax_sample1.owl

Throws:

java.lang.Exception - All Exceptions.

edu.ucsf.qpaca.parser

Class TestKGMLParsing

java.lang.Object

└ junit.framework.Assert

└ junit.framework.TestCase

└ edu.ucsf.qpaca.parser.TestKGMLParsing

All Implemented Interfaces:

junit.framework.Test

```
public class TestKGMLParsing
```

```
extends junit.framework.TestCase
```

Tests the KGML-QPACA mapper.

Constructor Summary

[TestKGMLParsing\(\)](#)

Method Summary

void	testMapper1()
	Tests the KGML mapper using a sample KGML file located at sampleData/hsa00062.xml

Methods inherited from class junit.framework.TestCase

countTestCases, createResult, getName, run, run, runBare, runTest, setName, setUp, tearDown, toString

Methods inherited from class junit.framework.Assert

```
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertFalse, assertFalse, assertNotNull, assertNotNull,
assertNotSame, assertNotSame, assertNull, assertNull, assertEquals,
assertEquals, assertEquals, assertEquals, fail, fail
```

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,
wait, wait
```

Constructor Detail

TestKGMLParsing

```
public TestKGMLParsing()
```

Method Detail

testMapper1

```
public void testMapper1()
    throws java.lang.Exception
```

Tests the KGML mapper using a sample KGML file located at sampleData/hsa00062.xml

Throws:

```
java.lang.Exception
```

edu.ucsf.qpaca.parser
Class TestKGMLUtil

```
java.lang.Object
├─ junit.framework.Assert
│   └─ junit.framework.TestCase
│       └─ edu.ucsf.qpaca.parser.TestKGMLUtil
```

All Implemented Interfaces:
junit.framework.Test

```
public class TestKGMLUtil
```

```
extends junit.framework.TestCase
```

Tests the KGMLUtil Class.

Constructor Summary

[TestKGMLUtil\(\)](#)

Method Summary

void	testUtil() Tests the KGML Utility Class using a sample KGML file located at sampleData/ hsa00190.xml
------	--

Methods inherited from class junit.framework.TestCase

countTestCases, createResult, getName, run, run, runBare, runTest, setName, setUp, tearDown, toString

Methods inherited from class junit.framework.Assert

```
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertFalse, assertFalse, assertNotNull, assertNotNull,
assertNotSame, assertNotSame, assertNull, assertNull, assertEquals,
assertEquals, assertEquals, assertEquals, fail, fail
```

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,
wait, wait
```

Constructor Detail

TestKGMLUtil

```
public TestKGMLUtil()
```

Method Detail

testUtil

```
public void testUtil()
    throws java.lang.Exception
```

Tests the KGML Utility Class using a sample KGML file located at sampleData/hsa00190.xml

Throws:

java.lang.Exception - All Exceptions.

edu.ucsf.qpaca.parser
Class TestParsing

```
java.lang.Object
├─ junit.framework.Assert
│   └─ junit.framework.TestCase
│       └─ edu.ucsf.qpaca.parser.TestParsing
```

All Implemented Interfaces:
junit.framework.Test

```
public class TestParsing
```

```
extends junit.framework.TestCase
```

```
Tests the BioPAX-QPACA mapper.
```

Constructor Summary

[TestParsing\(\)](#)

Method Summary

void	testComplexMapping() Tests that we can map BioPAX Complexes Correctly.
void	testMapper1() Tests the Mapper on a valid BioPAX file located at sampleData/biopax_sample1.owl

Methods inherited from class junit.framework.TestCase

countTestCases, createResult, getName, run, run, runBare, runTest, setName, setUp, tearDown, toString

Methods inherited from class junit.framework.Assert

```
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertFalse, assertFalse, assertNotNull, assertNotNull,
assertNotSame, assertNotSame, assertNull, assertNull, assertEquals,
assertSame, assertTrue, assertTrue, fail, fail
```

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,
wait, wait
```

Constructor Detail

TestParsing

```
public TestParsing()
```

Method Detail

testComplexMapping

```
public void testComplexMapping()
    throws java.lang.Exception
```

Tests that we can map BioPAX Complexes Correctly.

Throws:

java.lang.Exception - All Exceptions.

testMapper1

```
public void testMapper1()
    throws java.lang.Exception
```

Tests the Mapper on a valid BioPAX file located at sampleData/biopax_sample1.owl

Throws:

`java.lang.Exception` - All Exceptions.



edu.ucsf.qpaca.parser
Class TestPathLang

```
java.lang.Object
├── junit.framework.Assert
│   └── junit.framework.TestCase
│       └── edu.ucsf.qpaca.parser.TestPathLang
```

All Implemented Interfaces:
junit.framework.Test

```
public class TestPathLang
```

```
extends junit.framework.TestCase
```

Tests the QPACA language parser.

Constructor Summary

[TestPathLang\(\)](#)

Method Summary

void	testUtil() Tests the QPACA language parser using a valid QPACA file located at sampleData/ RTK.current.qpaca
------	--

Methods inherited from class junit.framework.TestCase

countTestCases, createResult, getName, run, run, runBare, runTest, setName, setUp, tearDown, toString

Methods inherited from class junit.framework.Assert

```
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertEquals, assertEquals, assertEquals, assertEquals, assertEquals,
assertFalse, assertFalse, assertNotNull, assertNotNull,
assertNotSame, assertNotSame, assertNull, assertNull, assertEquals,
assertEquals, assertEquals, assertEquals, fail, fail
```

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,
wait, wait
```

Constructor Detail

TestPathLang

```
public TestPathLang()
```

Method Detail

testUtil

```
public void testUtil()
    throws java.lang.Exception
```

Tests the QPACA language parser using a valid QPACA file located at sampleData/RTK.current.qpaca

Throws:

java.lang.Exception - All Exceptions.

edu.ucsf.qpaca.parser.util

Class BioPaxConstants

java.lang.Object

└─ edu.ucsf.qpaca.parser.util.BioPaxConstants

public class **BioPaxConstants**

extends java.lang.Object

BioPAX Constants.

Field Summary	
static java. lang.String	BIOCHEMICAL REACTION BioPAX Class: biochemicalReaction
static org. jdom. Namespace	BIOPAX_LEVEL_1_NAMESPACE BioPAX Level 1 Namespace.
static java. lang.String	BIOPAX_LEVEL_1_NAMESPACE_URI BioPAX Level 1 Namespace URI.
static org. jdom. Namespace	BIOPAX_LEVEL_2_NAMESPACE BioPAX Level 2 Namespace.
static java. lang.String	BIOPAX_LEVEL_2_NAMESPACE_URI BioPAX Level 2 Namespace URI.
static java. lang.String	BIOPAX_NAMESPACE_PREFIX BioPAX Namespace Prefix.
static java. lang.String	CATALYSIS BioPAX Class: catalysis
static java. lang.String	COMPLEX BioPAX Class: complex.
static java. lang.String	COMPLEX_ASSEMBLY BioPAX Class: complexAssembly

static java. lang.String	<u>CONTROL</u> BioPAX Class: control
static java. lang.String	<u>CONTROL_TYPE_ACTIVATION</u> Control Type: ACTIVATION
static java. lang.String	<u>CONTROL_TYPE_ACTIVATION_ALLOSTERIC</u> Control Type: ACTIVATION-ALLOSTERIC
static java. lang.String	<u>CONTROL_TYPE_ACTIVATION_NONALLOSTERIC</u> Control Type: ACTIVATION-NONALLOSTERIC
static java. lang.String	<u>CONTROL_TYPE_ACTIVATION_UNKMECH</u> Control Type: ACTIVATION-UNKMECH
static java. lang.String	<u>CONTROL_TYPE_INHIBITION</u> Control Type: INHIBITION
static java. lang.String	<u>CONTROL_TYPE_INHIBITION_ALLOSTERIC</u> Control Type: INHIBITION-ALLOSTERIC
static java. lang.String	<u>CONTROL_TYPE_INHIBITION_COMPETITIVE</u> Control Type: INHIBITION-COMPETITIVE
static java. lang.String	<u>CONTROL_TYPE_INHIBITION_IRREVERSIBLE</u> Control Type: INHIBITION-IRREVERSIBLE
static java. lang.String	<u>CONTROL_TYPE_INHIBITION_NONCOMPETITIVE</u> Control Type: INHIBITION-NONCOMPETITIVE
static java. lang.String	<u>CONTROL_TYPE_INHIBITION_OTHER</u> Control Type: INHIBITION-OTHER
static java. lang.String	<u>CONTROL_TYPE_INHIBITION_UNCOMPETITIVE</u> Control Type: INHIBITION-UNCOMPETITIVE
static java. lang.String	<u>CONTROL_TYPE_INHIBITION_UNKMECH</u> Control Type: INHIBITION-UNKMECH
static java. lang.String	<u>CONVERSION</u> BioPAX Class: conversion
static java. lang.String	<u>DNA</u> BioPAX Class: dna
static java. lang.String	<u>INTERACTION</u> BioPAX Class: interaction.
static java. lang.String	<u>MODULATION</u> BioPAX Class: modulation
static java. lang.String	<u>PATHWAY</u> BioPAX Class: pathway
static java. lang.String	<u>PHYSICAL_ENTITY</u> BioPAX Class: physicalEntity.

static java. lang.String	PHYSICAL INTERACTION BioPAX Class: physicalInteraction.
static java. lang.String	PROTEIN BioPAX Class: protein
static java. lang.String	RNA BioPAX Class: rna
static java. lang.String	SMALL MOLECULE BioPAX Class: smallMolecule.
static java. lang.String	TRANSPORT BioPAX Class: transport
static java. lang.String	TRANSPORT WITH BIOCHEMICAL REACTION BioPAX Class: transportWithBiochemicalReaction
static java. lang.String	UXREF BioPAX Class: UnificationXref

Constructor Summary

[BioPaxConstants](#) ()
Constructor.

Method Summary

java.util.Set	getInteractionSet () Gets a Set of all Interaction Entity Names.
java.util.Set	getPhysicalEntitySet () Gets a Set of all Physical Entity Names
static boolean	isActivation (java.lang.String controlType)
boolean	isControlInteraction (java.lang.String elementName) Determines if the Specified Element is of type: control.
boolean	isConversionInteraction (java.lang.String elementName) Determines if the Specified Element is of type: conversion.
static boolean	isInhibition (java.lang.String controlType)
boolean	isInteraction (java.lang.String elementName) Determines if the Specified Element is of type: interaction.
boolean	isPathway (java.lang.String elementName) Determines if the Specified Element is of type: pathway.
boolean	isPhysicalEntity (java.lang.String elementName) Determines if the Specified Element is of type: physical entity.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

BIOCHEMICAL_REACTION

```
public static final java.lang.String BIOCHEMICAL_REACTION
```

BioPAX Class: biochemicalReaction

See Also:

[Constant Field Values](#)

BIOPAX_LEVEL_1_NAMESPACE

```
public static final org.jdom.Namespace BIOPAX_LEVEL_1_NAMESPACE
```

BioPAX Level 1 Namespace.

BIOPAX_LEVEL_1_NAMESPACE_URI

```
public static final java.lang.String BIOPAX_LEVEL_1_NAMESPACE_URI
```

BioPAX Level 1 Namespace URI.

See Also:

[Constant Field Values](#)

BIOPAX_LEVEL_2_NAMESPACE

```
public static final org.jdom.Namespace BIOPAX_LEVEL_2_NAMESPACE
```

BioPAX Level 2 Namespace.

BIOPAX_LEVEL_2_NAMESPACE_URI

```
public static final java.lang.String BIOPAX_LEVEL_2_NAMESPACE_URI
```

BioPAX Level 2 Namespace URI.

See Also:

[Constant Field Values](#)

BIOPAX_NAMESPACE_PREFIX

```
public static final java.lang.String BIOPAX_NAMESPACE_PREFIX
```

BioPAX Namespace Prefix.

See Also:

[Constant Field Values](#)

CATALYSIS

```
public static final java.lang.String CATALYSIS
```

BioPAX Class: catalysis

See Also:

[Constant Field Values](#)

COMPLEX

```
public static final java.lang.String COMPLEX
```

BioPAX Class: complex.

See Also:

[Constant Field Values](#)

COMPLEX_ASSEMBLY

```
public static final java.lang.String COMPLEX_ASSEMBLY
```

BioPAX Class: complexAssembly

See Also:

[Constant Field Values](#)

CONTROL

```
public static final java.lang.String CONTROL
```

BioPAX Class: control

See Also:

[Constant Field Values](#)

CONTROL_TYPE_ACTIVATION

```
public static final java.lang.String CONTROL_TYPE_ACTIVATION
```

Control Type: ACTIVATION

See Also:

[Constant Field Values](#)

CONTROL_TYPE_ACTIVATION_ALLOSTERIC

```
public static final java.lang.String CONTROL_TYPE_ACTIVATION_ALLOSTERIC
```

Control Type: ACTIVATION-ALLOSTERIC

See Also:

[Constant Field Values](#)

CONTROL_TYPE_ACTIVATION_NONALLOSTERIC

```
public static final java.lang.String CONTROL_TYPE_ACTIVATION_NONALLOSTERIC
```

Control Type: ACTIVATION-NONALLOSTERIC

See Also:

[Constant Field Values](#)

CONTROL_TYPE_ACTIVATION_UNKMECH

```
public static final java.lang.String CONTROL_TYPE_ACTIVATION_UNKMECH
```

Control Type: ACTIVATION-UNKMECH

See Also:

[Constant Field Values](#)

CONTROL_TYPE_INHIBITION

```
public static final java.lang.String CONTROL_TYPE_INHIBITION
```

Control Type: INHIBITION

See Also:

[Constant Field Values](#)

CONTROL_TYPE_INHIBITION_ALLOSTERIC

```
public static final java.lang.String CONTROL_TYPE_INHIBITION_ALLOSTERIC
```

Control Type: INHIBITION-ALLOSTERIC

See Also:

[Constant Field Values](#)

CONTROL_TYPE_INHIBITION_COMPETITIVE

```
public static final java.lang.String CONTROL_TYPE_INHIBITION_COMPETITIVE
```

Control Type: INHIBITION-COMPETITIVE

See Also:

[Constant Field Values](#)

CONTROL_TYPE_INHIBITION_IRREVERSIBLE

```
public static final java.lang.String CONTROL_TYPE_INHIBITION_IRREVERSIBLE
```

Control Type: INHIBITION-IRREVERSIBLE

See Also:

[Constant Field Values](#)

CONTROL_TYPE_INHIBITION_NONCOMPETITIVE

```
public static final java.lang.String CONTROL_TYPE_INHIBITION_NONCOMPETITIVE
```

Control Type: INHIBITION-NONCOMPETITIVE

See Also:

[Constant Field Values](#)

CONTROL_TYPE_INHIBITION_OTHER

```
public static final java.lang.String CONTROL_TYPE_INHIBITION_OTHER
```

Control Type: INHIBITION-OTHER

See Also:

[Constant Field Values](#)

CONTROL_TYPE_INHIBITION_UNCOMPETITIVE

```
public static final java.lang.String CONTROL_TYPE_INHIBITION_UNCOMPETITIVE
```

Control Type: INHIBITION-UNCOMPETITIVE

See Also:

[Constant Field Values](#)

CONTROL_TYPE_INHIBITION_UNKMECH

```
public static final java.lang.String CONTROL_TYPE_INHIBITION_UNKMECH
```

Control Type: INHIBITION-UNKMECH

See Also:

[Constant Field Values](#)

CONVERSION

```
public static final java.lang.String CONVERSION
```

BioPAX Class: conversion

See Also:

[Constant Field Values](#)

DNA

public static final java.lang.String **DNA**

BioPAX Class: dna

See Also:

[Constant Field Values](#)

INTERACTION

public static final java.lang.String **INTERACTION**

BioPAX Class: interaction.

See Also:

[Constant Field Values](#)

MODULATION

public static final java.lang.String **MODULATION**

BioPAX Class: modulation

See Also:

[Constant Field Values](#)

PATHWAY

public static final java.lang.String **PATHWAY**

BioPAX Class: pathway

See Also:

PHYSICAL_ENTITY

```
public static final java.lang.String PHYSICAL_ENTITY
```

BioPAX Class: physicalEntity.

See Also:

[Constant Field Values](#)

PHYSICAL_INTERACTION

```
public static final java.lang.String PHYSICAL_INTERACTION
```

BioPAX Class: physicalInteraction.

See Also:

[Constant Field Values](#)

PROTEIN

```
public static final java.lang.String PROTEIN
```

BioPAX Class: protein

See Also:

[Constant Field Values](#)

RNA

```
public static final java.lang.String RNA
```

BioPAX Class: rna

See Also:

[Constant Field Values](#)

SMALL_MOLECULE

```
public static final java.lang.String SMALL_MOLECULE
```

BioPAX Class: smallMolecule.

See Also:

[Constant Field Values](#)

TRANSPORT

```
public static final java.lang.String TRANSPORT
```

BioPAX Class: transport

See Also:

[Constant Field Values](#)

TRANSPORT_WITH_BIOCHEMICAL_REACTION

```
public static final java.lang.String TRANSPORT_WITH_BIOCHEMICAL_REACTION
```

BioPAX Class: transportWithBiochemicalReaction

See Also:

[Constant Field Values](#)

UXREF

```
public static final java.lang.String UXREF
```

BioPAX Class: UnificationXref

See Also:

[Constant Field Values](#)

Constructor Detail

BioPaxConstants

```
public BioPaxConstants()
```

Constructor.

Method Detail

getInteractionSet

```
public java.util.Set getInteractionSet()
```

Gets a Set of all Interaction Entity Names.

Returns:

Set of Strings.

getPhysicalEntitySet

```
public java.util.Set getPhysicalEntitySet()
```

Gets a Set of all Physical Entity Names

Returns:

Set of Strings

isActivation

```
public static boolean isActivation(java.lang.String controlType)
```

isControlInteraction

```
public boolean isControlInteraction(java.lang.String elementName)
```

Determines if the Specified Element is of type: control.

Parameters:

elementName - Element Name.

Returns:

boolean value.

isConversionInteraction

```
public boolean isConversionInteraction(java.lang.String elementName)
```

Determines if the Specified Element is of type: conversion.

Parameters:

elementName - Element Name.

Returns:

boolean value.

isInhibition

```
public static boolean isInhibition(java.lang.String controlType)
```

isInteraction

```
public boolean isInteraction(java.lang.String elementName)
```

Determines if the Specified Element is of type: interaction.

Parameters:

elementName - Element Name.

Returns:

boolean value.

isPathway

```
public boolean isPathway(java.lang.String elementName)
```

Determines if the Specified Element is of type: pathway.

Parameters:

elementName - Element Name.

Returns:

boolean value.

isPhysicalEntity

```
public boolean isPhysicalEntity(java.lang.String elementName)
```

Determines if the Specified Element is of type: physical entity.

Parameters:

elementName - Element Name.

Returns:

boolean value.

edu.ucsf.qpaca.parser.util

Class BioPaxUtil

java.lang.Object

└─ edu.ucsf.qpaca.parser.util.BioPaxUtil

```
public class BioPaxUtil
```

```
extends java.lang.Object
```

BioPax Utility Class.

Constructor Summary

[BioPaxUtil](#)(java.io.Reader reader)
Constructor.

Method Summary

java.util.ArrayList	getComplexList () Gets List of PhysicalEntity/complex Resources.
java.util.ArrayList	getInteractionList () Gets List of Interaction Resources.
java.util.ArrayList	getPathwayList () Gets list of Pathway Resources.
java.util.ArrayList	getPhysicalEntityList () Gets List of Physical Entity Resources.
java.util.HashMap	getRdfResourceMap () Gets HashMap of All RDF Resources, keyed by RDF ID.
org.jdom.Element	getRootElement () Gets the Root Element of the BioPAX Tree.

<pre>java. util. ArrayList</pre>	<pre>getTopLevelComponentList()</pre> <p>Gets a List of all Pathways, Interactions, and Physical Entities.</p>
----------------------------------	--

<p>Methods inherited from class java.lang.Object</p> <pre>clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait</pre>

Constructor Detail

BioPaxUtil

```
public BioPaxUtil(java.io.Reader reader)
    throws java.io.IOException,
           org.jdom.JDOMException,
           BioPaxException
```

Constructor.

Parameters:

reader - Reader Object.

Throws:

java.io.IOException - Input/Output Error.
 org.jdom.JDOMException - XML Error.
[BioPaxException](#)

Method Detail

getComplexList

```
public java.util.ArrayList getComplexList()
```

Gets List of PhysicalEntity/complex Resources.

Returns:

ArrayList of JDOM Element Objects.

getInteractionList

```
public java.util.ArrayList getInteractionList()
```

Gets List of Interaction Resources.

Returns:
ArrayList of JDOM Element Objects.

getPathwayList

```
public java.util.ArrayList getPathwayList()
```

Gets list of Pathway Resources.

Returns:
ArrayList of JDOM Element Objects.

getPhysicalEntityList

```
public java.util.ArrayList getPhysicalEntityList()
```

Gets List of Physical Entity Resources.

Returns:
ArrayList of JDOM Element Objects.

getRdfResourceMap

```
public java.util.HashMap getRdfResourceMap()
```

Gets HashMap of All RDF Resources, keyed by RDF ID.

Returns:
HashMap of All RDF Resources, keyed by RDF ID.

getRootElement

```
public org.jdom.Element getRootElement()
```

Gets the Root Element of the BioPAX Tree.

Returns:

JDOM Element Object.

getTopLevelComponentList

```
public java.util.ArrayList getTopLevelComponentList()
```

Gets a List of all Pathways, Interactions, and Physical Entities.

Returns:

ArrayList of JDOM Element Objects.

edu.ucsf.qpaca.parser.util

Class KGMLConstants

java.lang.Object

└─ edu.ucsf.qpaca.parser.util.KGMLConstants

public class **KGMLConstants**

extends java.lang.Object

KGML Constants.

Field Summary	
static java. lang.String	ACTIVATION relation subtype: activation
static java. lang.String	BINDING ASSOCIATION relation subtype: binding/association
static java. lang.String	COMPLEX entry type: complex of gene products
static java. lang.String	COMPOUND relation subtype: compound
static java. lang.String	DEPHOSPHORYLATION relation subtype: dephosphorylation
static java. lang.String	DISSOCIATION relation subtype: dissociation
static java. lang.String	ECREL relationship type: enzyme-enzyme.
static java. lang.String	ENZYME entry type: enzyme.
static java. lang.String	EXPRESSION relation subtype: expression

static java. lang.String	<u>GENE</u> entry type: gene
static java. lang.String	<u>GEREL</u> relationship type: gene expression (transcription factor-gene)
static java. lang.String	<u>GLYCOSYLATION</u> relation subtype: glycosylation
static java. lang.String	<u>HIDDEN_COMPOUND</u> relation subtype: hidden compound
static java. lang.String	<u>INDIRECT_EFFECT</u> Control Type: indirect effect
static java. lang.String	<u>INHIBITION</u> relation subtype: inhibition
static java. lang.String	<u>IRREVERSIBLE</u> reaction type: irreversible
static java. lang.String	<u>MAPLINK</u> relationship type: link to another pathway
static java. lang.String	<u>METHYLATION</u> relation subtype: methylation
static java. lang.String	<u>ORTHOLOG</u> entry type: ortholog.
static java. lang.String	<u>PATHWAY</u> entry type: linked pathway.
static java. lang.String	<u>PCREL</u> relationship type: protein-compound
static java. lang.String	<u>PHOSPHORYLATION</u> relation subtype: phosphorylation
static java. lang.String	<u>PPREL</u> relationship type: protein-protein.
static java. lang.String	<u>REPRESSION</u> relation subtype: repression
static java. lang.String	<u>REVERSIBLE</u> reaction type: reversible
static java. lang.String	<u>SMALL_MOLECULE</u> entry type: chemical compound
static java. lang.String	<u>STATE_CHANGE</u> relation subtype: state change
static java. lang.String	<u>UBIQUINATION</u> relation subtype: ubiquination

Constructor Summary

[KGMLConstants](#)()

Initializes the constants.

Method Summary

static boolean	isActivation (java.lang.String subtype) Determines if an interaction is of an activation type
----------------	--

static boolean	isInhibition (java.lang.String subtype) Determines if an interaction is of an inhibition type
----------------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

ACTIVATION

```
public static final java.lang.String ACTIVATION
```

relation subtype: activation

See Also:

[Constant Field Values](#)

BINDING_ASSOCIATION

```
public static final java.lang.String BINDING_ASSOCIATION
```

relation subtype: binding/association

See Also:

[Constant Field Values](#)

COMPLEX

public static final java.lang.String **COMPLEX**

entry type: complex of gene products

See Also:

[Constant Field Values](#)

COMPOUND

public static final java.lang.String **COMPOUND**

relation subtype: compound

See Also:

[Constant Field Values](#)

DEPHOSPHORYLATION

public static final java.lang.String **DEPHOSPHORYLATION**

relation subtype: dephosphorylation

See Also:

[Constant Field Values](#)

DISSOCIATION

public static final java.lang.String **DISSOCIATION**

relation subtype: dissociation

See Also:

[Constant Field Values](#)

ECREL

```
public static final java.lang.String ECREL
```

relationship type: enzyme-enzyme.

See Also:

[Constant Field Values](#)

ENZYME

```
public static final java.lang.String ENZYME
```

entry type: enzyme.

See Also:

[Constant Field Values](#)

EXPRESSION

```
public static final java.lang.String EXPRESSION
```

relation subtype: expression

See Also:

[Constant Field Values](#)

GENE

```
public static final java.lang.String GENE
```

entry type: gene

See Also:

[Constant Field Values](#)

GEREL

```
public static final java.lang.String GEREL
```

relationship type: gene expression (transcription factor-gene)

See Also:

[Constant Field Values](#)

GLYCOSYLATION

```
public static final java.lang.String GLYCOSYLATION
```

relation subtype: glycosylation

See Also:

[Constant Field Values](#)

HIDDEN_COMPOUND

```
public static final java.lang.String HIDDEN_COMPOUND
```

relation subtype: hidden compound

See Also:

[Constant Field Values](#)

INDIRECT_EFFECT

```
public static final java.lang.String INDIRECT_EFFECT
```

Control Type: indirect effect

See Also:

[Constant Field Values](#)

INHIBITION

```
public static final java.lang.String INHIBITION
```

relation subtype: inhibition

See Also:

[Constant Field Values](#)

IRREVERSIBLE

```
public static final java.lang.String IRREVERSIBLE
```

reaction type: irreversible

See Also:

[Constant Field Values](#)

MAPLINK

```
public static final java.lang.String MAPLINK
```

relationship type: link to another pathway

See Also:

[Constant Field Values](#)

METHYLATION

public static final java.lang.String **METHYLATION**

relation subtype: methylation

See Also:

[Constant Field Values](#)

ORTHOLOG

public static final java.lang.String **ORTHOLOG**

entry type: ortholog.

See Also:

[Constant Field Values](#)

PATHWAY

public static final java.lang.String **PATHWAY**

entry type: linked pathway.

See Also:

[Constant Field Values](#)

PCREL

public static final java.lang.String **PCREL**

relationship type: protein-compound

See Also:

[Constant Field Values](#)

PHOSPHORYLATION

```
public static final java.lang.String PHOSPHORYLATION
```

relation subtype: phosphorylation

See Also:

[Constant Field Values](#)

PPREL

```
public static final java.lang.String PPREL
```

relationship type: protein-protein.

See Also:

[Constant Field Values](#)

REPRESSION

```
public static final java.lang.String REPRESSION
```

relation subtype: repression

See Also:

[Constant Field Values](#)

REVERSIBLE

public static final java.lang.String **REVERSIBLE**

reaction type: reversible

See Also:

[Constant Field Values](#)

SMALL_MOLECULE

public static final java.lang.String **SMALL_MOLECULE**

entry type: chemical compound

See Also:

[Constant Field Values](#)

STATE_CHANGE

public static final java.lang.String **STATE_CHANGE**

relation subtype: state change

See Also:

[Constant Field Values](#)

UBIQUINATION

public static final java.lang.String **UBIQUINATION**

relation subtype: ubiquination

See Also:

[Constant Field Values](#)

Constructor Detail

KGMLConstants

```
public KGMLConstants()
```

Initializes the constants.

Method Detail

isActivation

```
public static boolean isActivation(java.lang.String subtype)
```

Determines if an interaction is of an activation type

Parameters:

subtype - the type of interaction to evaluate

Returns:

true if the subtype represents activation, false otherwise.

isInhibition

```
public static boolean isInhibition(java.lang.String subtype)
```

Determines if an interaction is of an inhibition type

Parameters:

subtype - the type of interaction to evaluate

Returns:

true if the subtype represents inhibition, false otherwise.

edu.ucsf.qpaca.parser.util
Class KGMLUtil

java.lang.Object
└─ edu.ucsf.qpaca.parser.util.KGMLUtil

public class **KGMLUtil**

extends java.lang.Object

KEGG KGML Utility Class.

Constructor Summary

[KGMLUtil](#)(java.io.Reader reader)
Loads the KGML file.

Method Summary

java.util.HashMap	getAllEntriesMap () Retrieves a list of all entries.
java.util.TreeMap	getComplexMap () Retrieves a list of complexes.
java.util.HashMap	getEntityMap () Retrieves a list of Entities.
java.lang.String	getPathwayID () Retrieves the pathway name.
java.lang.String	getPathwayLabel () Retrieves the pathway label.
java.util.ArrayList	getReactionList () Retrieves a list of Reactions.

java. util. ArrayList	getRelationList() Retrieves a list of Relations.
org. jdom. Element	getRootElement() Retrieves the Root Element of the KGML Tree.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

KGMLUtil

```
public KGMLUtil(java.io.Reader reader)
    throws java.io.IOException,
           org.jdom.JDOMException,
           java.lang.Exception
```

Loads the KGML file.

Parameters:

reader - Reader Object.

Throws:

java.io.IOException - Input/Output Error.
org.jdom.JDOMException - XML Error.
java.lang.Exception

Method Detail

getAllEntriesMap

```
public java.util.HashMap getAllEntriesMap()
```

Retrieves a list of all entries.

Returns:

ArrayList of JDOM Element Objects.

getComplexMap

```
public java.util.TreeMap getComplexMap()
```

Retrieves a list of complexes.

Returns:
ArrayList of JDOM Element Objects.

getEntityMap

```
public java.util.HashMap getEntityMap()
```

Retrieves a list of Entities.

Returns:
ArrayList of JDOM Element Objects.

getPathwayID

```
public java.lang.String getPathwayID()
```

Retrieves the pathway name.

Returns:
pathway name

getPathwayLabel

```
public java.lang.String getPathwayLabel()
```

Retrieves the pathway label.

Returns:

pathway label

getReactionList

```
public java.util.ArrayList getReactionList()
```

Retrieves a list of Reactions.

Returns:

ArrayList of JDOM Element Objects.

getRelationList

```
public java.util.ArrayList getRelationList()
```

Retrieves a list of Relations.

Returns:

ArrayList of JDOM Element Objects.

getRootElement

```
public org.jdom.Element getRootElement()
```

Retrieves the Root Element of the KGML Tree.

Returns:

JDOM Element Object.

edu.ucsf.qpaca.parser.util
Class OwlConstants

java.lang.Object
└─ edu.ucsf.qpaca.parser.util.OwlConstants

public class **OwlConstants**

extends java.lang.Object

OWL (Web Ontology Language) Constants.

Field Summary	
static java. lang.String	OWL_IMPORTS_ELEMENT OWL Imports Element Name
static org. jdom. Namespace	OWL_NAMESPACE RDF Namespace Object.
static java. lang.String	OWL_NAMESPACE_PREFIX OWL Namespace Prefix
static java. lang.String	OWL_NAMESPACE_URI OWL Namespace URI.
static java. lang.String	OWL_ONTOLOGY_ELEMENT OWL Ontology Element

Constructor Summary
OwlConstants()

Method Summary

Methods inherited from class java.lang.Object
--

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll,  
toString, wait, wait, wait
```

Field Detail

OWL_IMPORTS_ELEMENT

```
public static final java.lang.String OWL_IMPORTS_ELEMENT
```

OWL Imports Element Name

See Also:

[Constant Field Values](#)

OWL_NAMESPACE

```
public static final org.jdom.Namespace OWL_NAMESPACE
```

RDF Namespace Object.

OWL_NAMESPACE_PREFIX

```
public static final java.lang.String OWL_NAMESPACE_PREFIX
```

OWL Namespace Prefix

See Also:

[Constant Field Values](#)

OWL_NAMESPACE_URI

```
public static final java.lang.String OWL_NAMESPACE_URI
```

OWL Namespace URI.

See Also:

[Constant Field Values](#)

OWL_ONTOLOGY_ELEMENT

```
public static final java.lang.String OWL_ONTOLOGY_ELEMENT
```

OWL Ontology Element

See Also:

[Constant Field Values](#)

Constructor Detail

OwlConstants

```
public OwlConstants()
```

edu.ucsf.qpaca.parser.util

Class QPACAConstants

java.lang.Object

└─ edu.ucsf.qpaca.parser.util.QPACAConstants

```
public class QPACAConstants
```

```
extends java.lang.Object
```

QPACA Constants.

Field Summary	
static java.lang.String	ACTIVATION PathwayEdge type: activation.
static java.lang.String	ALIAS PathwayAssembly type for families
static java.lang.String	COMPLEX PathwayAssembly type for complexes
static java.lang.String	COMPOUND PathwayAssembly type for complexes
static java.lang.String	CONVERSION PathwayEdge type: conversion.
static java.lang.String	DNA PathwayElement type: dna
static java.lang.String	EVENT PathwayVertex type: event
static java.lang.String	FAMILY PathwayAssembly type for families
static java.lang.String	GENE_PRODUCT PathwayElement type: gene product
static java.lang.String	INHIBITION PathwayEdge type: inhibition.

static java.lang.String	PHYSICAL ENTITY PathwayVertex type: physicalEntity
static java.lang.String	PROCESS PathwayElement type: process
static java.lang.String	PROTEIN PathwayElement type: protein
static java.lang.String	RNA PathwayElement type: rna
static java.lang.String	SINGLE PathwayAssembly type for assemblies containing a single item.
static java.lang.String	SMALL MOLECULE PathwayElement type: small molecule

Constructor Summary	
QPACConstants()	Constructor.

Method Summary	
static boolean	isControlInteraction (java.lang.String type) Determines if the type specifies a control interaction
static boolean	isConversionInteraction (java.lang.String type) Determines if the type specifies a conversion interaction
static boolean	isNotSmallMolecule (java.lang.String type) Determines if the type specifies a dna/rna/protein/gene_product PathwayElement

Methods inherited from class java.lang.Object	
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	

Field Detail

ACTIVATION

```
public static final java.lang.String ACTIVATION
```

PathwayEdge type: activation.

See Also:

ALIAS

```
public static final java.lang.String ALIAS
```

PathwayAssembly type for families

See Also:

[Constant Field Values](#)

COMPLEX

```
public static final java.lang.String COMPLEX
```

PathwayAssembly type for complexes

See Also:

[Constant Field Values](#)

COMPOUND

```
public static final java.lang.String COMPOUND
```

PathwayAssembly type for complexes

See Also:

[Constant Field Values](#)

CONVERSION

```
public static final java.lang.String CONVERSION
```

PathwayEdge type: conversion.

See Also:

[Constant Field Values](#)

DNA

```
public static final java.lang.String DNA
```

PathwayElement type: dna

See Also:

[Constant Field Values](#)

EVENT

```
public static final java.lang.String EVENT
```

PathwayVertex type: event

See Also:

[Constant Field Values](#)

FAMILY

```
public static final java.lang.String FAMILY
```

PathwayAssembly type for families

See Also:

[Constant Field Values](#)

GENE_PRODUCT

```
public static final java.lang.String GENE_PRODUCT
```

PathwayElement type: gene product

See Also:

[Constant Field Values](#)

INHIBITION

```
public static final java.lang.String INHIBITION
```

PathwayEdge type: inhibition.

See Also:

[Constant Field Values](#)

PHYSICAL_ENTITY

```
public static final java.lang.String PHYSICAL_ENTITY
```

PathwayVertex type: physicalEntity

See Also:

[Constant Field Values](#)

PROCESS

```
public static final java.lang.String PROCESS
```

PathwayElement type: process

See Also:

[Constant Field Values](#)

PROTEIN

```
public static final java.lang.String PROTEIN
```

PathwayElement type: protein

See Also:

[Constant Field Values](#)

RNA

```
public static final java.lang.String RNA
```

PathwayElement type: rna

See Also:

[Constant Field Values](#)

SINGLE

```
public static final java.lang.String SINGLE
```

PathwayAssembly type for assemblies containing a single item.

See Also:

[Constant Field Values](#)

SMALL_MOLECULE

```
public static final java.lang.String SMALL_MOLECULE
```

PathwayElement type: small molecule

See Also:

[Constant Field Values](#)

Constructor Detail

QPACAConstants

```
public QPACAConstants()
```

Constructor.

Method Detail

isControlInteraction

```
public static final boolean isControlInteraction(java.lang.String type)
```

Determines if the type specifies a control interaction

Parameters:

type - type.

Returns:

boolean value.

isConversionInteraction

```
public static final boolean isConversionInteraction(java.lang.String type)
```

Determines if the type specifies a conversion interaction

Parameters:

type - type.

Returns:

boolean value.

isNotSmallMolecule

```
public static final boolean isNotSmallMolecule(java.lang.String type)
```

Determines if the type specifies a dna/rna/protein/gene_product PathwayElement

Parameters:

type - type.

Returns:
boolean value.



edu.ucsf.qpaca.parser.util
Class QPACAUtil

java.lang.Object
└─ edu.ucsf.qpaca.parser.util.QPACAUtil

public class **QPACAUtil**

extends java.lang.Object

QPACA Pathway language Utility Class.

Constructor Summary

[QPACAUtil](#)(java.io.Reader reader)
Initializes the parser.

Method Summary

java.util.HashMap	getAllEntriesMap () Retrieves a list of all entries.
java.util.TreeMap	getComplexMap () Retrieves a list of complexes.
java.util.HashMap	getEntityMap () Retrieves a list of Entities.
Pathway	getPathway ()
java.lang.String	getPathwayID () Retrieves the pathway name.
java.lang.String	getPathwayLabel () Retrieves the pathway label.

java. util. ArrayList	getRelationList() Retrieves a list of Relations.
-----------------------------	---

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

QPACAUtil

```
public QPACAUtil(java.io.Reader reader)
    throws java.io.IOException,
           java.lang.Exception
```

Initializes the parser.

Parameters:

reader - Reader Object.

Throws:

java.io.IOException - Input/Output Error.
java.lang.Exception

Method Detail

getAllEntriesMap

```
public java.util.HashMap getAllEntriesMap()
```

Retrieves a list of all entries.

Returns:

ArrayList of JDOM Element Objects.

getComplexMap

```
public java.util.TreeMap getComplexMap()
```

Retrieves a list of complexes.

Returns:

ArrayList of JDOM Element Objects.

getEntityMap

```
public java.util.HashMap getEntityMap()
```

Retrieves a list of Entities.

Returns:

ArrayList of JDOM Element Objects.

getPathway

```
public Pathway getPathway()
```

getPathwayID

```
public java.lang.String getPathwayID()
```

Retrieves the pathway name.

Returns:

pathway name

getPathwayLabel

```
public java.lang.String getPathwayLabel()
```

Retrieves the pathway label.

Returns:
pathway label

getRelationList

```
public java.util.ArrayList getRelationList()
```

Retrieves a list of Relations.

Returns:
ArrayList of JDOM Element Objects.

edu.ucsf.qpaca.parser.util
Class RdfConstants

java.lang.Object
└─ edu.ucsf.qpaca.parser.util.RdfConstants

public class **RdfConstants**

extends java.lang.Object

RDF (Resource Description Framework) Constants.

Field Summary	
static java. lang.String	ABOUT_ATTRIBUTE RDF About Attribute
static java. lang.String	DATATYPE_ATTRIBUTE RDF Datatype Attribute
static java. lang.String	ID_ATTRIBUTE RDF ID Attribute
static org. jdom. Namespace	RDF_NAMESPACE RDF Namespace Object.
static java. lang.String	RDF_NAMESPACE_PREFIX RDF Namespace Prefix
static java. lang.String	RDF_NAMESPACE_URI RDF Namespace URI
static java. lang.String	RDF_ROOT_NAME RDF Root Name.
static java. lang.String	RESOURCE_ATTRIBUTE RDF Resource Attribute

Constructor Summary

[RdfConstants](#)()

Method Summary

Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`,
`toString`, `wait`, `wait`, `wait`

Field Detail

ABOUT_ATTRIBUTE

```
public static final java.lang.String ABOUT_ATTRIBUTE
```

RDF About Attribute

See Also:

[Constant Field Values](#)

DATATYPE_ATTRIBUTE

```
public static final java.lang.String DATATYPE_ATTRIBUTE
```

RDF Datatype Attribute

See Also:

[Constant Field Values](#)

ID_ATTRIBUTE

```
public static final java.lang.String ID_ATTRIBUTE
```

RDF ID Attribute

See Also:

[Constant Field Values](#)

RDF_NAMESPACE

```
public static final org.jdom.Namespace RDF_NAMESPACE
```

RDF Namespace Object.

RDF_NAMESPACE_PREFIX

```
public static final java.lang.String RDF_NAMESPACE_PREFIX
```

RDF Namespace Prefix

See Also:

[Constant Field Values](#)

RDF_NAMESPACE_URI

```
public static final java.lang.String RDF_NAMESPACE_URI
```

RDF Namespace URI

See Also:

[Constant Field Values](#)

RDF_ROOT_NAME

```
public static final java.lang.String RDF_ROOT_NAME
```

RDF Root Name.

See Also:

[Constant Field Values](#)

RESOURCE_ATTRIBUTE

```
public static final java.lang.String RESOURCE_ATTRIBUTE
```

RDF Resource Attribute

See Also:

[Constant Field Values](#)

Constructor Detail

RdfConstants

```
public RdfConstants()
```

edu.ucsf.qpaca.parser.util

Class RdfQuery

java.lang.Object

└─ edu.ucsf.qpaca.parser.util.RdfQuery

```
public class RdfQuery
```

```
extends java.lang.Object
```

Enables XPath-"lite" Queries on RDF Documents.

To make things really simple, this implementation ignore case, and ignores all Namespaces.

Constructor Summary

[RdfQuery](#)(java.util.HashMap rdfMap)

A Hashmap of all XML Elements, keyed by RDF ID.

Method Summary

org.jdom.Element	getNode (org.jdom.Element e, java.lang.String query) Gets First Node that match the XPath-"lite" Query.
java.util.ArrayList	getNodes (org.jdom.Element e, java.lang.String query) Gets all Nodes that match the XPath-"lite" Query.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

RdfQuery

```
public RdfQuery(java.util.HashMap rdfMap)
```

A Hashmap of all XML Elements, keyed by RDF ID.

Parameters:

rdfMap - HashMap of RDF ID to XML Element.

Method Detail

getNode

```
public org.jdom.Element getNode(org.jdom.Element e,  
                                java.lang.String query)
```

Gets First Node that match the XPath-"lite" Query.

Parameters:

e - Target Element.
query - XPath-"lite" Query.

Returns:

ArrayList of JDOM Elements, which match the query.

getNodes

```
public java.util.ArrayList getNodes(org.jdom.Element e,  
                                     java.lang.String query)
```

Gets all Nodes that match the XPath-"lite" Query.

Parameters:

e - Target Element.
query - XPath-"lite" Query.

Returns:

ArrayList of JDOM Elements, which match the query.

edu.ucsf.qpaca.parser.util

Class RdfUtil

java.lang.Object

└─ edu.ucsf.qpaca.parser.util.RdfUtil

```
public class RdfUtil
```

```
extends java.lang.Object
```

Misc RDF Utilities.

Constructor Summary

[RdfUtil\(\)](#)

Method Summary

static java. lang.String	removeHashMark (java.lang.String referenceId) Strips out leading hash mark #, if necessary.
-----------------------------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

RdfUtil

```
public RdfUtil()
```

Method Detail

removeHashMark

```
public static java.lang.String removeHashMark(java.lang.String referenceId)
```

Strips out leading hash mark #, if necessary.

Parameters:

referenceId - String with leading hash mark.

Returns:

String without hashmark.

edu.ucsf.qpaca.parser.util

Class BioPaxException

java.lang.Object

└─ java.lang.Throwable

└─ java.lang.Exception

└─ edu.ucsf.qpaca.parser.util.BioPaxException

All Implemented Interfaces:

java.io.Serializable

```
public class BioPaxException
```

```
extends java.lang.Exception
```

Exception thrown to indicate a syntax error in a BioPAX file.

See Also:

[Serialized Form](#)

Constructor Summary

[BioPaxException](#)()

[BioPaxException](#)(java.lang.String msg)

Method Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

BioPaxException

```
public BioPaxException()
```

BioPaxException

```
public BioPaxException(java.lang.String msg)
```

edu.ucsf.qpaca.parser.util

Class KGMLException

```
java.lang.Object
├── java.lang.Throwable
│   └── java.lang.Exception
│       └── edu.ucsf.qpaca.parser.util.KGMLException
```

All Implemented Interfaces:

java.io.Serializable

```
public class KGMLException
```

```
extends java.lang.Exception
```

Exception thrown to indicate a syntax error in a KGML file.

See Also:

[Serialized Form](#)

Constructor Summary

[KGMLException](#)()

[KGMLException](#)(java.lang.String msg)

Method Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

KGMLException

```
public KGMLException()
```

KGMLException

```
public KGMLException(java.lang.String msg)
```

edu.ucsf.qpaca.parser.util

Class QPACAEException

java.lang.Object

└─ java.lang.Throwable

└─ java.lang.Exception

└─ edu.ucsf.qpaca.parser.util.QPACAEException

All Implemented Interfaces:

java.io.Serializable

```
public class QPACAEException
```

extends java.lang.Exception

Exception thrown to indicate a syntax error in a QPACA file.

See Also:

[Serialized Form](#)

Constructor Summary

[QPACAEException](#)()

[QPACAEException](#)(java.lang.String msg)

Method Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

Constructor Detail

QPACAEException

```
public QPACAEException()
```

QPACAEException

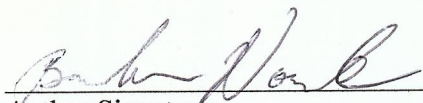
```
public QPACAEException(java.lang.String msg)
```

UCSF Library Release

Publishing Agreement

It is the policy of the University to encourage the distribution of all theses and dissertations. Copies of all UCSF theses and dissertations will be routed to the library via the Graduate Division. The library will make all theses and dissertations accessible to the public and will preserve these to the best of their abilities, in perpetuity.

I hereby grant permission to the Graduate Division of the University of California, San Francisco to release copies of my thesis or dissertation to the Campus Library to provide access and preservation, in whole or in part, in perpetuity.



Author Signature

June 6, 2007
Date