Entwicklung von Methoden zur automatischen Generierung, grafischen Darstellung und interaktiven Analyse von metabolischen Netzwerken

I n a u g u r a l - D i s s e r t a t i o n

zur

Erlangung des Doktorgrades

der Mathematisch-Naturwissenschaftlichen Fakultät

der Universität zu Köln

vorgelegt von

Ralph Oliver Schunk

aus Meerbusch

Köln

2006

Berichterstatter/-in: Prof. Dr. Dietmar Schomburg
Prof. Dr. Michael Jünger

Tag der letzten mündlichen Prüfung: Montag, der 12. Februar 2007

Die Dissertation wurde mit dem Textsatzsystem LaTeX 2_ε formatiert.

Für meine Eltern

Danksagung

An dieser Stelle möchte ich mich bei Herrn Prof. Dr. Dietmar Schomburg für die Vergabe und Betreuung dieser sehr interessanten Dissertation bedanken. Er selbst und die von ihm geleitete Arbeitsgruppe „Metabolomforschung“ ermöglichten stets eine angenehme und produktive Arbeitsatmosphäre.

Weiterer Dank gilt ebenfalls

den Arbeitsgruppen von Herrn Prof. Dr. Michael Jünger und Frau Prof. Dr. Petra Mutzel, für die überaus kompetente Beratung und Hilfestellung zu allen Fragen rund um das automatische Zeichnen von Graphen;

allen Angehörigen des Cologne University BioInformatics Center (CUBIC) und des Instituts für Biochemie für ihre freundliche Diskussionsbereitschaft und ihre sehr hilfreiche Kooperation.

Ein sehr herzlicher Dank gilt Bärbel & Arthur Kapitz und Martin Rausch, die mir beim Korrekturlesen dieser Arbeit geholfen haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Zusammenfassung

Die große Anzahl der metabolischen Reaktionen und der an ihnen beteiligten Komponenten kann nur mit Methoden zur automatisierten Darstellung und Analyse von metabolischen Netzwerken erforscht werden. Mit dem CUBIC Pathway Editor „*Cupe*“ existiert jetzt ein Programm, das auf einzigartige Art und Weise die Generierung, Darstellung und Analyse von Stoffwechselnetzwerken mit den aktuellsten Methoden zum automatischen Zeichnen von Graphen verbindet. Dank eines integrierten Datenbestandes von ca. 32.000 Reaktionen und ca. 47.000 Komponenten bietet *Cupe* zudem die größte Reaktionsdatenbank unter allen bisher bekannten Programmen für die Stoffwechselanalyse. Der Anwender wird sowohl bei der manuellen als auch bei der automatischen Erzeugung von metabolischen Netzwerken unterstützt und hat mit *Cupe* die Möglichkeit, alle Netzwerkelemente völlig frei zu formatieren. Besonders die Fähigkeit, von *Cupe* Subnetzwerke in Superknoten bzw. Clustern zusammenzufassen und einen Metabolitpool zu verwalten, erlaubt es, den Graphen eines Reaktionsnetzwerks erheblich zu vereinfachen. Dadurch ist es möglich geworden, die Lesbarkeit automatisch gezeichneter Graphen deutlich zu verbessern. Mit der Entwicklung der *Cubic-Sparse-Matrix* verfügt *Cupe* über ein neuartiges und effizientes Datenmodell. Auf der Grundlage dieses Datenmodells konnte die *„Mengenlehre für metabolische Netzwerke“* für die vergleichende Analyse von Reaktionsnetzwerken entwickelt werden. Weitere Analysemethoden können über die einfach zu bedienende Erweiterungsschnittstelle zu *Cupe* hinzugefügt werden. Die verschiedenen Module von *Cupe* verknüpfen so zahlreiche Forschungsgebiete und bilden dadurch eine interdisziplinäre Forschungs- und Ausbildungsplattform, deren Weiterentwicklung über das im Internet bereitgestellte *Cupe Knowledge Portal* koordiniert wird.

Abstract

The large number of metabolic reactions and components involved therein can only be investigated by using methods for automated representation and analysis of metabolic networks. The new CUBIC Pathway Editor *"Cupe"* uniquely combines the generation, representation and analysis of metabolic networks with the most current methods for automatic graph drawing. With its integrated metabolic data set of approx. 32 000 reactions and approx. 47 000 components, *Cupe* offers the largest reaction database of all known programs for metabolic analysis. The *Cupe* user is being supported during the manual as well as during the automatic creation of metabolic networks and due to *Cupe* has the possibility to completely freely format all network elements. *Cupe* is able to summarize sub networks in superknots respectively clusters and is able to maintain a metabolite pool. These abilities in particular allow *Cupe* to substantially simplify the graph of a reaction network. It thus became possible to clearly improve the legibility of automatically drawn graphs. With the development of the *Cubic-Sparse-Matrix*, *Cupe* has a novel and efficient data model at its disposal. Based on this data model the *"set theory for metabolic networks"* has been developed. It is being used for the comparative analysis of reaction networks. An easy to use plug-in interface allows the addition of further analysis methods to *Cupe*. The different modules of *Cupe* thus link numerous scientific fields of research and thereby build up an interdisciplinary research and training platform. Its further development will be coordinated through the internet-based *"Cupe Knowledge Portal"*.

1 Einleitung

Aus der Fülle der metabolischen Reaktionen (ca. 32.000) und der an ihnen beteiligten Komponenten (ca. 47.000) in der Brendadatenbank (Stand von 06.2006; Schomburg *et al.*, 2000) lässt sich die Notwendigkeit einer automatisierten Darstellung und Analyse von metabolischen Netzwerken offensichtlich ableiten. In dieser Dissertation wurde das Programm *Cubic Pathway Editor* (*Cupe*) entwickelt, das metabolische Netzwerke automatisch darstellen und analysieren kann. Zusätzlich ist es möglich, Netzwerke interaktiv zu editieren. Zunächst werden die Anforderungen an grafische Editoren für metabolische Netzwerke dargestellt und anschließend die bisher entwickelten Programme kurz besprochen. Eine vergleichende Übersicht bietet Tabelle 1.1 auf Seite 19. Im Text sind die Schlüsselwörter fett gedruckt, für die in der Tabelle eigene Zeilen vorhanden sind.

1.1 Anforderungen an einen grafischen Editor für metabolische Netzwerke

Die notwendigen Eigenschaften eines Editors können in die folgenden vier Gruppen zusammengefasst werden: die Benutzerschnittstelle, die Fähigkeiten der automatischen Graphen-Darstellung, die Möglichkeiten zur manuellen Formatierung und Interaktion sowie die Zusammenfassung aller Methoden zur Analyse, Simulation und Vernetzung der Metabolismen.

1.1.1 Benutzerschnittstelle

Die grafische Benutzerschnittstelle (GUI) muss eine zusammenfassende Darstellung aller notwendigen Informationen ermöglichen. Dieses betrifft in erster Linie natürlich den Graphen des metabolischen Netzwerkes. Darüber hinaus sind aber auch weitere Informationen wie etwa die Strukturformeln oder Verknüpfungen zu PDB-Datenbankeinträgen (Berman *et al.*, 2000) denkbar. Dabei gilt, je umfangreicher die dargestellten Informationen sind, um so mehr Zeit benötigt die Darstellung über ein **Web-Interface**. Das gilt auch, wenn die Daten lokal verfügbar sind, da der geschwindigkeitsbestimmende Schritt das Rendern der Darstellung im Browser ist (z.B. ExPASy, Michal, 1999; Brandenburg *et al.*, 2001). Hier kann die Arbeitsgeschwindigkeit durch die Entwicklung spezialisierter **Editoren** deutlich gesteigert werden (z.B. Demir *et al.*, 2002; Trost *et al.*, 2002). Aus Gründen der Portabilität wird dabei oft Java eingesetzt (z.B. Junker *et al.*, 2006; Shannon *et al.*, 2003). Bei großen Graphen kann die Arbeitsgeschwindigkeit – bedingt durch die Java-Implementation –

allerdings sehr langsam sein (wie z.B. bei Šīrava *et al.*, 2002). Hier erscheint es sinnvoll, auf Programmier-Bibliotheken wie etwa QT (Trolltech, 2003) und/oder LEDA (Mehlhorn & Näher, 1989; Näher & Mehlhorn, 1990a) zurückzugreifen, um so Portabilität und hohe Verarbeitungsgeschwindigkeit besser zu vereinen (vergleiche Küffner *et al.*, 2000).

1.1.2 Automatisches Layout

> „Graph Drawing is the best possible field I can think of: It merges aesthetics, mathematical beauty and wonderful algorithms. It therefore provides a harmonic balance between the left and right brain parts.“
> D. E. Knuth (GD' 1996)

Für die automatische Generierung der Graphen von metabolischen Netzwerken sind folgende Punkte zu beachten:

Lesbarkeit

Die Lesbarkeit von automatisch generierten Graphen ist eines der Hauptanliegen von Informatikern, die sich mit „Graph Drawing“ beschäftigen. Die Formulierung von Kriterien, die die Lesbarkeit beeinflussen und ihre Implementation ist Teil der gegenwärtigen Forschung (z.B. AGD Alberts *et al.* (1997)). Neben primitiven Kriterien, wie z.B. die Reduzierung von Überkreuzungen der Kanten (vergleiche z.B. Algorithmic Solutions, 2001), wird dabei den weitaus komplexeren Kriterien, etwa der Darstellung der Symmetrie eines Netzwerkes, eine hohe Bedeutung zugesprochen (vergleiche Mutzel *et al.*, 2001).

Mentale Karte

Insbesondere bei der manuellen Interaktion über eine grafische Benutzeroberfläche muss es möglich sein, das bisherige Layout strukturell zu erhalten. Andernfalls wird bei jeder Veränderung eine komplette Neuorientierung in der Zeichnung nötig.

Nebenmetabolite

Die Anzahl der Knoten und Kanten nimmt deutlich zu und somit die Lesbarkeit einer Zeichnung stark ab, wenn Metabolite von Neben-Reaktionen (z.B. ATP-Verbrauch[1]) als gleichberechtigte Knoten behandelt werden. Es ist daher nötig, für solche Reaktionen eine gesonderte Darstellung zu bieten.

Gruppierung

Bei metabolischen Netzwerken ist es oft sinnvoll, sie in Subnetzwerke zu unterteilen (z.B. alle Pfade der Gluconeogenese). Auch sollten Stoffwechselprozesse bestimmter

[1] A+B+ATP $\rightleftharpoons$ C+D+ADP

Zellkompartimente nicht mit denen anderer Kompartimente vermischt werden (z.B. die Stoffwechselwege der Glykolyse im Cytoplasma mit der Atmungskette in den Mitochondrien). Daher muss es möglich sein, bestimmte Verbindungen und Reaktionswege in einem Cluster zu gruppieren. Verbindungen einer solchen Gruppe sollten dann beim automatischen Graphenlayout gruppiert bleiben und nicht über die Zeichenfläche verstreut werden. Im abstrakten Sinne würden solche Cluster einfach „große“ Knoten darstellen.

Fokussierung

Für die intuitive Erfassung der zweidimensionalen Darstellung eines metabolischen Netzwerkes ist es erforderlich, dass der Detailreichtum einer metabolischen Karte, mit zunehmender Entfernung vom Fokus der Betrachtung sukzessive abnimmt (vergleiche z.B. Karp *et al.*, 2002).

Runtime-Layout

Wenn ein Algorithmus zum Graphenlayout eine besonders schnelle Implementierung besitzt, so ist es möglich, das Graphenlayout nahezu simultan zur Benutzereingabe zu berechnen. Im Ergebnis führt das dann dazu, dass der Graph automatisch auf interaktive Positionsveränderungen einzelner Knoten reagiert. Für kleine Graphen und einer einfachen, hierarchischen Formatierung findet man dieses Runtime-Layout z.B. bei daVinci (Fröhlich, 1998) verwirklicht.

1.1.3 Manuelle Formatierung

Immer dann, wenn vorprozessierte Daten manuell analysiert oder nachbearbeitet werden, ist es notwendig, die manuellen Änderungen hervorzuheben. Der Anwender möchte die **Form von Kanten und Knoten** verändern, die **Farbe und Strichart/-stärke** beeinflussen, **Beschriftungen** anbringen, **neue Reaktionspfade** eintragen und gegebenenfalls **Teile des Graphenlayouts** verändern (vergleiche Hucka *et al.*, 2002b; Demir *et al.*, 2002; Trost *et al.*, 2002, usw.).

1.1.4 Analyse-Methoden

Bei der Untersuchung eines metabolischen (Sub-)Netzwerkes ist es schnell erforderlich, zirkuläre Systeme oder Teilbereiche mit besonderer Dichte, Umsatzraten usw. als **Subgraph** zu identifizieren, darzustellen und weiter zu analysieren.
Auch möchte man bei komplexen Systemen die **Reaktionspfade** identifizieren, die von einem Ausgangsstoff zu einem Endprodukt führen. Dabei sind natürlich Umsatzraten und „Nebenwege“ nur zwei Kriterien, die eine solche Darstellung beeinflussen (vergleiche Goesmann *et al.*, 2002).
Neben experimentell erhaltenen Informationen über Reaktionswege möchte man beispielsweise auch den Einfluss hypothetischer Reaktionspfade auf Charakteristika

des Reaktionssystems abschätzen oder von den experimentellen Daten auf *in vivo* Verhalten extrapolieren. Die dafür notwendigen **Simulationen und Analysemethoden** sollten dabei über eine wohldefinierte Schnittstelle mit der Graphen-Darstellung interagieren. Auf diese Weise können unterschiedliche Methoden von verschiedenen Forschungseinrichtungen in die Analyse der metabolischen Netzwerke eingebunden werden und das ohne besondere Kenntnisse über die Implementation und Repräsentation im Editor (z.B. Petrinetze oder Flussanalysen; Küffner *et al.*, 2000; Hucka *et al.*, 2002b; Pfeiffer *et al.*, 1999; Šīrava *et al.*, 2002).
Wenn man die Stoffwechselsysteme der eigenen Forschung analysiert, ist man natürlich auch schnell an **Vergleichen** mit bereits bestehenden Systemen interessiert. Dadurch wäre es z.B. möglich, bisher fehlende Reaktionswege zu identifizieren (siehe z.B. Kanehisa & Goto, 1999).
Auf der Grundlage bestehender Datenbanken und selbst eingepflegter Datenbestände ist es wünschenswert, dass bei der interaktiven Eingabe von metabolischen Pfaden solche, die sich von den Referenzdaten unterscheiden, zur Laufzeit direkt gekennzeichnet werden. So kann der interessierte Molekularbiologe bei der Aufarbeitung seiner Ergebnisse direkt auf Inkonsistenzen aufmerksam gemacht werden und bekommt die Möglichkeit, seine eigenen Daten zu **verifizieren**.
Um Umsetzungen von Verbindungen nachzuvollziehen, ist es sinnvoll, zusätzlich zu den Namen der an den Reaktionen beteiligten **Molekülen** auch deren **Summenformel** und **Lewis-Struktur** anzuzeigen (vergleiche Karp *et al.*, 2002; Brandenburg *et al.*, 2001; Kanehisa & Goto, 1999; Michal, 1999, usw.).
Die zur Verfügung stehenden molekularbiologisch fundierten Datenbanken liefern ein umfangreiches Angebot an zusätzlichen Informationen (z.B. Annotationen zu Enzymen, Enzymstrukturen, Wirkmechanismen usw.). Bei vielen dieser Informationen ist es wünschenswert, dass sie mit dem Graphen ihres metabolischen Systems **verknüpft** sind (Verknüpfungen mit anderen Datenbanken bestehen z.B. bei: Ellis *et al.*, 1998; Karp *et al.*, 2002, u.a.).

1.2 Methoden zur Darstellung und interaktiven Manipulation von metabolischen Netzwerken

Die bisher verfügbaren Möglichkeiten zum interaktiven Editieren und Analysieren metabolischer Pfade über eine grafische Benutzerschnittstelle haben unterschiedliche Fähigkeiten und sind unterschiedlichen Beschränkungen unterworfen. Die zur Zeit verfügbaren Methoden werden im Folgenden kurz vorgestellt. Eine vergleichende Übersicht gibt die Tabelle 1.1 auf Seite 19.

1.2.1 PathFinder

PathFinder ist eine Methode zur dynamischen Visualisierung von metabolischen Pfaden, als Grundlage dienen annotierte Genome (EMBL oder GenBank; Stösser *et al.*, 1999). Zu der Möglichkeit, eigene Datensätze zu benutzen, existiert noch eine

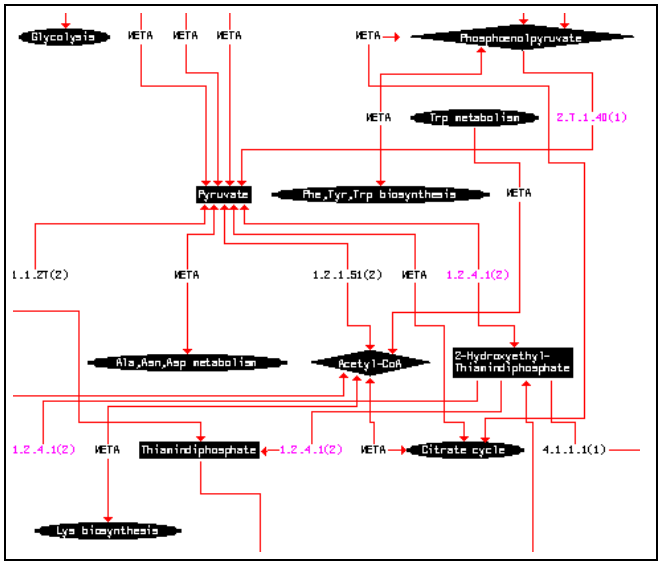


Abbildung 1.1: Layout von *PathFinder* Ausschnitt aus der Gluconeogenese. Die von VCG erzeugten Graphen sind sehr groß und lassen sich nicht editieren.

Sammlung von „Master Pathways“, die aus der KEGG-Datenbank (siehe Kanehisa & Goto, 1999) gewonnen wurden. Es existiert ein einfaches Web-Interface. Ein Editor wird zwar im Artikel erwähnt, er ist aber nicht verfügbar. Der Graph wird automatisch von VCG (siehe Sander, 1995) erzeugt und ist aufgrund langer Kanten und zahlreicher Überschneidungen unübersichtlich (siehe Abbildung 1.1). Es gibt keine Möglichkeit, Nebenmetabolite darzustellen sowie Graphenelemente zu gruppieren, fokussieren oder zu fixieren. Ebenfalls besteht keine Möglichkeit, das Format von Knoten und Kanten manuell zu verändern, Beschriftungen anzubringen oder das Graphenlayout zu beeinflussen. Es ist möglich, sich vorgefertigte Subgraphen anzusehen, Pfade zwischen zwei Metaboliten zu identifizieren und Reaktionswege mit eigenen Daten zu vergleichen. Kosten, wie etwa Gleichgewichtskonstanten oder stöchiometrische Koeffizienten, können den Kanten dabei nicht zugeordnet werden. Die Verknüpfung mit der Swiss-Prot Datenbank (Bairoch & Apweiler, 1998) ist unvollständig. Weitere Verknüpfungen mit anderen Datenbanken existieren nicht.

1.2.2 PathViewer

PathViewer ist Bestandteil des BioMiner-Projektes. Dabei handelt es sich um eine Sammlung von Programmen, die über ein einheitliches Datenmodell – BioCore – interagieren. PathViewer ist kein Editor, sondern kann lediglich die Reaktionspfade darstellen. Dabei bedient es sich zweier Layout-Methoden, um den Graphen zu formatieren. Die eine arbeitet hierarchisch die andere force-directed. Beide liefern unübersichtliche Ergebnisse, was wiederum an sehr langen Kanten, zahlreichen Überschneidungen und zusätzlichen Verknüpfungen zwischen den Kanten liegt (ver-

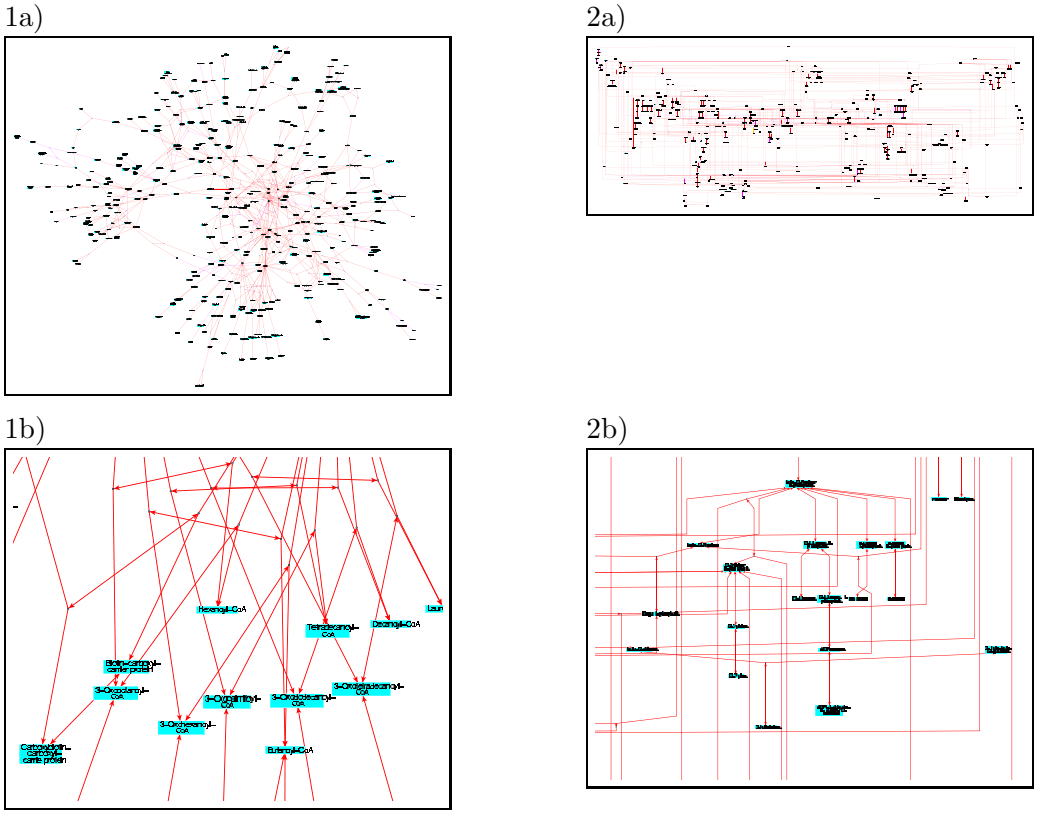


Abbildung 1.2: Graphenlayout von PathViewer Beispieldatensatz von der Internetseite zu PathViewer. 1a) Kräftebasiertes Layout, 1b) Ausschnitt, 2a) Hierarchisches Layout, 2b) Ausschnitt.

gleiche Abbildung 1.2). Die Knoten sind mit einer Metabolit-Liste verknüpft. Eine Sonderbehandlung von Nebenmetaboliten ist nicht möglich. PathViewer existiert als Java-Applikation und ist mit den Beispiel-Datensätzen sehr langsam.

1.2.3 DMD

Die Informationen über metabolische Pfade werden aus Datenbanken gewonnen (siehe Schomburg *et al.*, 2000; Bairoch, 1999; Selkov *et al.*, 1997, 1998; Overbeek *et al.*, 1999; Karp *et al.*, 1999, 1996; Kanehisa & Goto, 1999) und in Petri-Netze (Petri, 1962) übersetzt. Ausgehend von diesen vorprozessierten Daten, können dann „shortes path“-Analysen oder Vergleiche verschiedener Netzwerke durchgeführt und dargestellt werden. Die Algorithmen für das Graphenlayout wurden in Eigenleistung entwickelt. Die Zeichnung ist aufgrund ihrer hohen Dichte und der sich überlagernden Beschriftungen nahezu unlesbar (siehe Abbildung 1.3 auf der nächsten Seite). Es besteht keine Möglichkeit, die Graphen manuell zu editieren. Die über das WWW-Interface angebotene Möglichkeit, LEDA-Graphwin (Algorithmic Solutions,

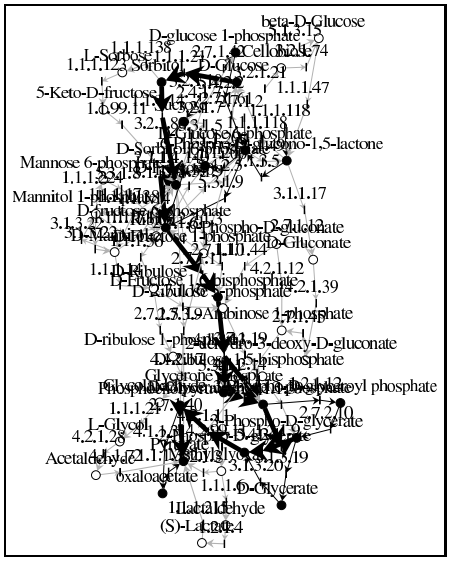


Abbildung 1.3: Automatischer Graph von DMD Die von DMD (Küffner *et al.*, 2000) erzeugten Graphen sind, aufgrund ihrer Dichte, im allgemeinem nicht lesbar und lassen sich nicht editieren.

2001) aufzurufen, um den Graphen manuell zu bearbeiten, funktioniert nicht richtig.

1.2.4 KEGG

Die KEGG-Datenbank ist im wesentlichen eine Sammlung von metabolischen Reaktionen. Aus dieser Sammlung werden auf überwiegend manueller Art und Weise metabolische Karten mit einer sehr guten Lesbarkeit erzeugt (vergleiche Abbildung 1.4 auf der nächsten Seite). Die über das Internet zugänglichen Karteninhalte sind untereinander und mit der EMBL-Datenbank verlinkt. Möglichkeiten zur interaktiven Manipulation oder vollautomatischen Generierung gibt es nicht.

1.2.5 Pathway Tools

Bei Pathway Tools handelt es sich um eine Sammlung von insgesamt drei Programm-Modulen: PathoLogic, Pathway/Genome-Navigator und -Editor. Die Module Navigator und Editor bieten eine gute Möglichkeit, metabolische Netzwerke zu analysieren. Mit Hilfe des Navigators kann man von einer allgeinen Gesamtübersicht zu immer detaillierteren Ansichten gelangen (siehe Abbildung 1.5 auf Seite 11). Die Komplexität des gesamten metabolischen Netzwerkes wird dabei durch Symbole verringert und vereinfacht. Aufgrund des hohen Abstraktionsgrads ist der Überblick, trotz seiner „Klarheit“, nicht intuitiv zu erfassen. Die Visualisierung erfolgt durch die Auswertung einer zugrunde liegenden Datenbank. Anwender, die ihre eigene Datenbanken pflegen, können diese über das Editor-Modul bearbeiten. Eine Möglichkeit,

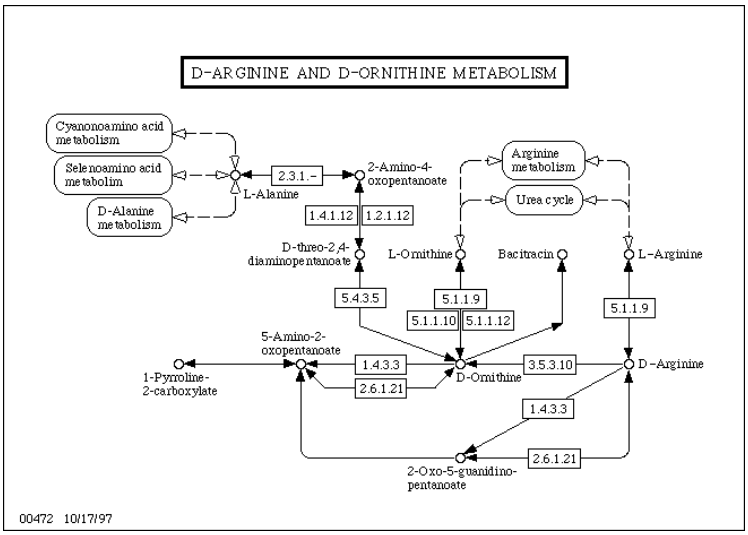


Abbildung 1.4: Pathway von KEGG Die von KEGG erzeugten Graphen werden überwiegend manuell erzeugt. Aufgrund dieses menschlichen Eingreifens sind die Karten sehr gut lesbar. Einfluss darauf, wie und wann metabolische Karten verändert werden, hat man jedoch nur indirekt.

den erzeugten Graphen direkt zu editieren bzw. formatieren, besteht nicht. Neue metabolische Reaktionen müssen erst in die Datenbank eingepflegt werden. Analysen sind insofern möglich, als dass Datensätze von verschiedenen Organismen miteinander verglichen werden können (siehe z.B. Paley & Karp, 2001). Simulationen, „shortes path"-Analysen usw. sind nicht möglich. Zusätzlich existieren zahlreiche Verknüpfungen zu weiteren Informationen (Lokalisation auf den Chromosomen, EC-Nummern, chemische Informationen usw.).

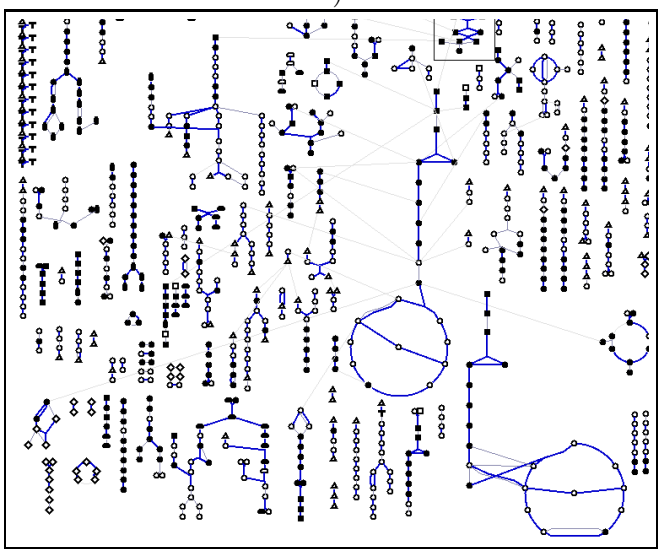
1.2.6 BioJAKE

BioJAKE ist für die manuelle Konstruktion von Graphen entwickelt worden und baut auf Graphviz (AT&T Research, 2003) auf. Zur Zeit ist es nicht öffentlich verfügbar. Es ist wahrscheinlich nicht mehr als eigenständiges Projekt vorhanden, sondern in PIES (Wong, 2001) aufgegangen. Das Programm ist in der Lage, Moleküle zu gruppieren und aus Datensätzen Graphen zu erzeugen. Das Layout des Graphen muss dabei allerdings manuell erfolgen. Analysemethoden sind nicht verfügbar. Es ist aber möglich, Graphen als Hintergrundbild zu laden. Dieses war ursprünglich als Schablone für eigene Graphen gedacht. Es ist dadurch aber auch möglich, Vergleiche anzustellen.

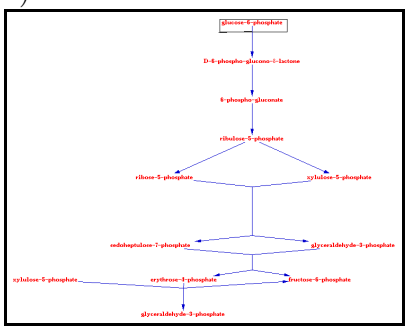
1.2.7 BioPath

Mittels *BioPath* kann man sich Übersichtsdiagramme von metabolischen Systemen ansehen (auf der Grundlage der Boehringer-Karte siehe Michal, 1999). Es ist

1)



2)



3)

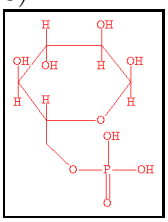
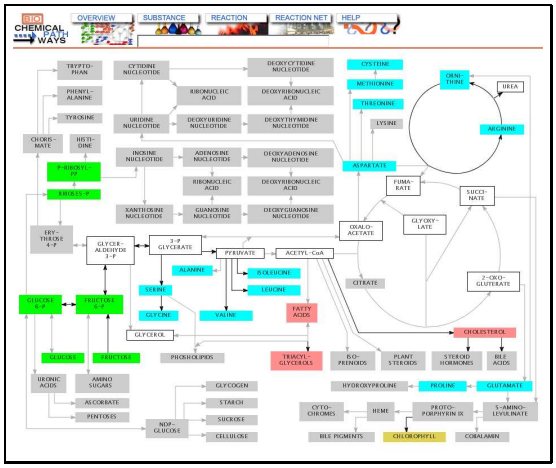


Abbildung 1.5: Der Navigator von Pathway Tools Ausgehend von einer allgemeinen Gesamtübersicht (1), kann man über weitere Schritte immer mehr Details betrachten (2 + 3). Zusätzlich existieren zahlreiche Verknüpfungen zu textbasierten Informationen.

a) 

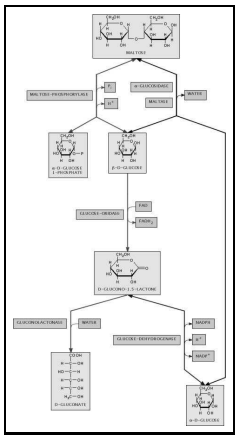
b) 

Abbildung 1.6: Web-Interface von *BioPath* a) Übersicht b) Beispielreaktion der nicht öffentlichen Webseite.

möglich, nach bestimmten Substanzen zu suchen, sich Reaktionswege zwischen zwei Substanzen darstellen zu lassen und dabei die Länge der Pfade vorzugeben (siehe Abbildung 1.6). Zur Zeit ist *BioPath*, aufgrund von offenen Copyright-Problemen, nicht öffentlich zugänglich. Über einen privaten Zugang konnte die Funktionalität nicht getestet werden, weil das Web-Interface auf Abfragen mit Fehlermeldungen reagiert. Weitergehende Analysen, manuelle Veränderungen und ein externer Editor existieren nicht. Die Graphen sollen mit weiteren Informationen verknüpft sein.

1.2.8 GeneNet

GeneNet ist eigentlich eine objektorientierte Datenbank physiologischer Prozesse, die eine Java-Anwendung für die automatische Visualisierung dieser Daten bereitstellt. Leider ist das Programm in wesentlichen Bereichen unvollständig und konnte daher kaum getestet werden. Hervorgehoben werden kann hier nur die Idee der Kompartiment-Unterteilung, vor deren Hintergrund die Abschnitte der Reaktionsgraphen, entsprechend ihrer Lokalisation in der Zelle, gezeichnet werden. Durch Filter können Graphen verschiedener Spezies überlagert und verglichen werden. Das Layout scheint auf einigen vorgegebenen Schablonen zu beruhen.

1.2.9 Patika

Patika ist auf der einen Seite ein Server mit Datenbank für zellulare Netzwerke und auf der anderen Seite ein Client mit grafischem Editor. Patika ist in der Lage, Graphen von metabolischen Prozessen zu erzeugen, sie zu bearbeiten und zu analysieren und wieder zurück in die Datenbank zu schreiben. Die Zeichnung wird dabei auf der Grundlage von spring embedder (Eades, 1984) automatisch erzeugt. Leider ist Patika zur Zeit nicht zu erhalten.

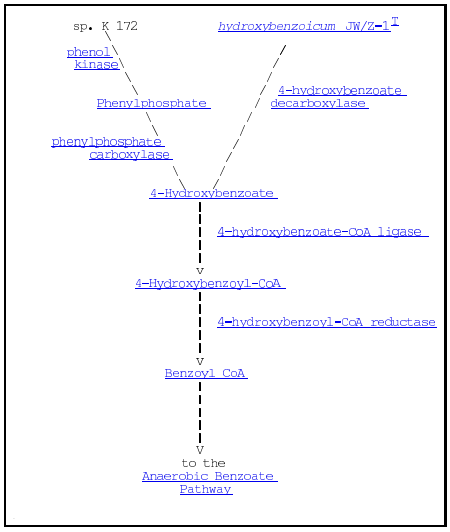


Abbildung 1.7: The University of Minnesota Biocatalysis/Biodegradation Database Diese textbasierte Darstellung wird dynamisch (siehe Ellis *et al.*, 1998) auf Anfrage generiert. Zusätzlich sind zu vielen Pathways auch handgezeichnete Grafiken verfügbar.

1.2.10 UM-BBD

Die University of Minnesota Biocatalysis/Biodegradation Database (UM-BBD) bietet eine Java-Applikation namens CORE an, mittels derer dynamisch metabolische Pfade für die Präsentation im Internet erzeugt werden können. Diese textbasierten „Abbildungen“, sind mit zahlreichen zusätzlichen Informationen verlinkt und können von jeder UM-BBD-Reaktion aus berechnet werden (siehe Abbildung 1.7). Sie zeichnen sich durch eine sehr gute Lesbarkeit aus. Komplexe Pfade werden dabei allerdings in mehrere html-Seiten aufgeteilt. Für viele Graphen werden zusätzlich manuell erstellte Karten angeboten (zumeist von `KEGG`, siehe Kanehisa & Goto, 1999). Manuelle Eingriffsmöglichkeiten und weitere Analysemethoden bestehen nicht.

1.2.11 GSCope

Bei GSCope handelt es sich um ein Programm-Modul von KnowledgeEditor, welcher für die Analyse von Microarray-Experimenten entwickelt wurde. Das Programm ist daher hauptsächlich darauf ausgerichtet, Expressionsprofile möglichen biologischen Stoffwechselwegen zuzuordnen. Ähnlich wie bei Hucka *et al.* (2002b), wird dazu ein Datenbestand in der Auszeichnungssprache XML[2] aufgezeichnet und bearbeitet. Für die Darstellung der so gewonnenen Netzwerke kommt dann GSCope zum Einsatz. Das Graphenlayout der Netzwerke erfolgt dabei von Hand (siehe Abbildung 1.8 auf der nächsten Seite). Die Graphen sind nicht mit weiterführenden Informationen ver-

[2]extended markup language

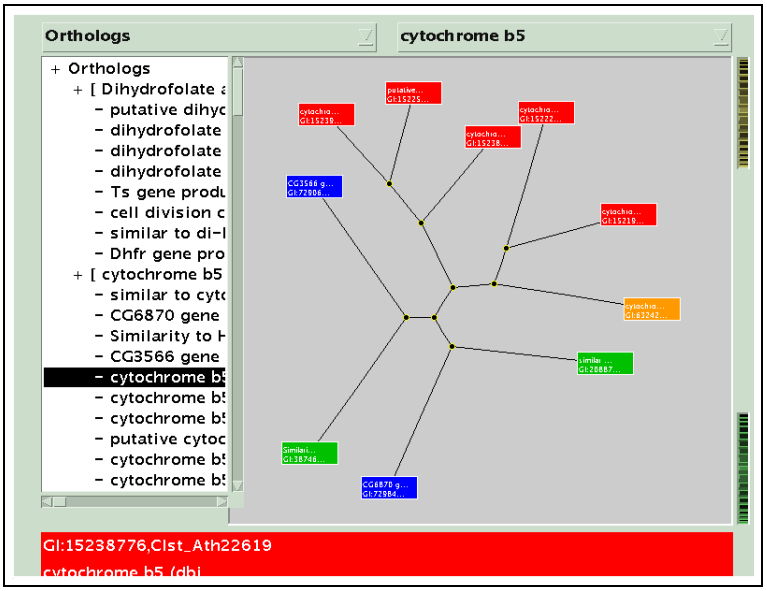


Abbildung 1.8: The GSCope Viewer GSCope ist nur zum Anzeigen von metabolischen Pfaden geeignet. Die Graphen sind von Hand gezeichnet.

knüpft. Einzige Analysemethode bleibt die Verknüpfung mit den Expressionsprofilen aus den Microarray-Experimenten.

1.2.12 ExPASy

ExPASy ist die in Michal (1999) abgedruckte Boehringer-Karte „Biochemical Pathways“ in digital aufgearbeiteter Form (siehe Abbildung 1.9 auf der nächsten Seite). Die sehr gute Lesbarkeit der Karte rührt von ihrer manuellen Erstellung her. Enzyme sind jeweils mit Datenbankeinträgen verlinkt (Hofmann *et al.*, 1999; Schomburg *et al.*, 2000; Overbeek *et al.*, 1999; Kanehisa & Goto, 1999; Bairoch & Apweiler, 1998; Overbeek *et al.*, 2000; Selkov *et al.*, 1996; NC-IUBMB, 1992; PubMed, 2003). Verbindungen sind nicht mit weiteren Informationen verlinkt. Einfluss auf die Darstellung hat man nicht.

1.2.13 Java editor for biological pathways

Der Java editor for biological pathways ist ein reines Editorprogramm ohne zusätzliche Funktionalität (Trost *et al.*, 2002). Mit Hilfe vorgefertigter Formen kann man schnell metabolische Prozesse mit einheitlichem Layout anfertigen. Die so entstandenen Karten sind dadurch prinzipiell mit denen von Kanehisa & Goto (1999) vergleichbar (siehe Abbildung 1.10 auf Seite 16). Der Anwender muss also eine gewisse Zeit in eine lesbare Graphenzeichnung investieren, ohne dass das Programm ihn dabei unterstützen kann. Die Graphen können nur im jpeg-Format gespeichert werden, was die Weiterverarbeitung in anderen Vektorgrafik-Programmen erschwert.

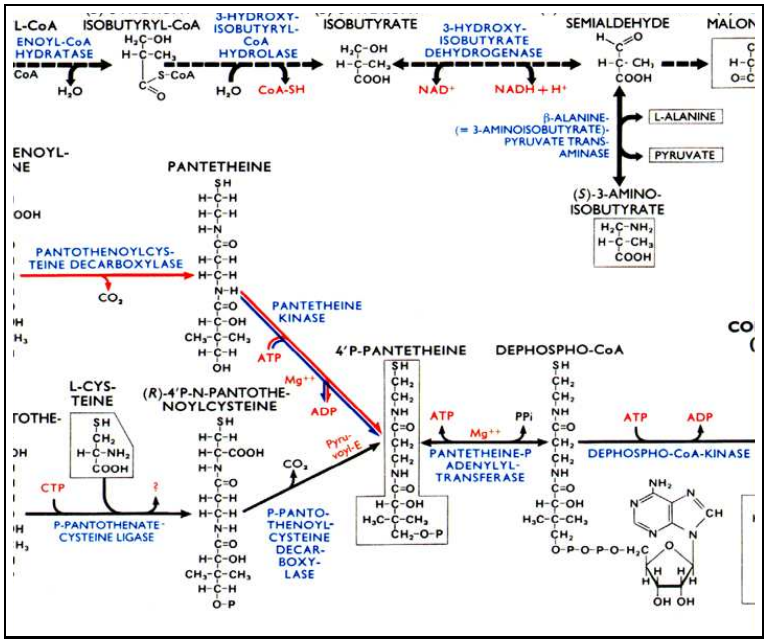


Abbildung 1.9: ExPASy: Die digitale Boehringer-Karte Bei ExPASy handelt es sich um eine handgezeichnete Karte, die mit zusätzlichen Verknüpfungen die Funktionalität der bekannten Boehringer-Karte erweitert.

1.2.14 JDesigner

Zusammen mit der Entwicklung von Systems Biology (Hucka *et al.*, 2001) werden zahlreiche Programme implementiert und unter Systems Biology Workbench Hucka *et al.* (2002a) zur Verfügung gestellt. JDesigner (siehe Abbildung 1.11 auf der nächsten Seite) ist ein Programm aus dieser Arbeit. Es handelt sich um ein Programm zur Darstellung und Generierung von metabolischen Karten. Ein automatisches Layout erfolgt nicht. Graphen von Netzwerken werden im SBML-Datenformat (Hucka *et al.*, 2002b) abgespeichert. Dieses Dateiformat kann von Jarnac gelesen werden und so bereits einige Simulationen auf den metabolischen Netzwerken durchgeführt werden. Die Anzahl der Programme, die SBML-Daten verarbeiten können, wird mit der Zeit weiter zunehmen. Der Nachfolger von JDesigner ist CellDesigner (siehe 1.12 auf Seite 17). Diese Version reimplementiert JDesigner in der Programmiersprache Java und erhöht so die Verfügbarkeit für unterschiedliche Betriebssysteme. Gleichzeitig sind der Editor und die Integration der Analysemethoden verbessert worden.

1.2.15 VANTED

VANTED (Junker *et al.*, 2006) ist ein Programm zum Anzeigen und Analysieren von Netzwerken und experimentellen Daten, die mit ihnen assoziiert sind. Auf einen im GML-Format (Himsolt, M., 1996) abgelegten Graphen eines Netzwerkes können

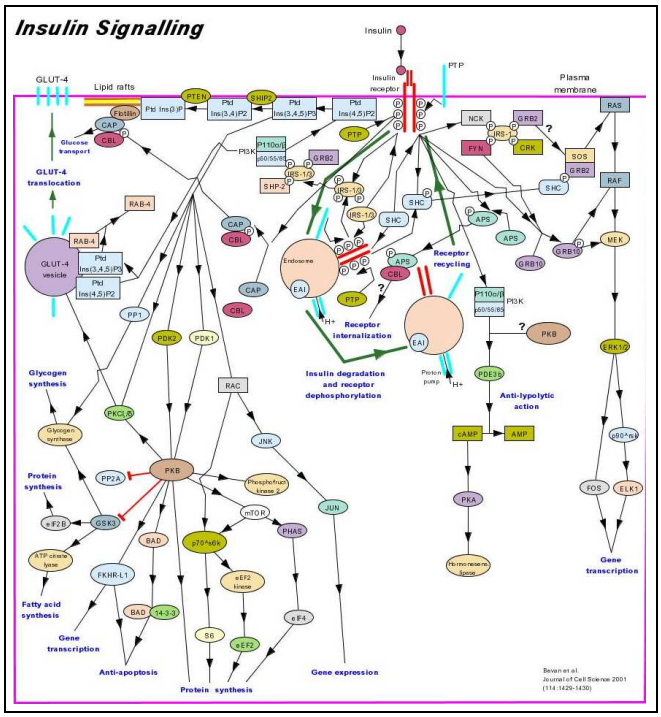


Abbildung 1.10: Pathways mit „Java editor for biological pathways“ Mit „Java editor for biological pathways“ steht ein Java-Programm zur Verfügung, mit dem man eigene Karten zeichnen kann. Dabei wird der Anwender jedoch weder beim Layout unterstützt noch stehen Analysemethoden zur Verfügung.

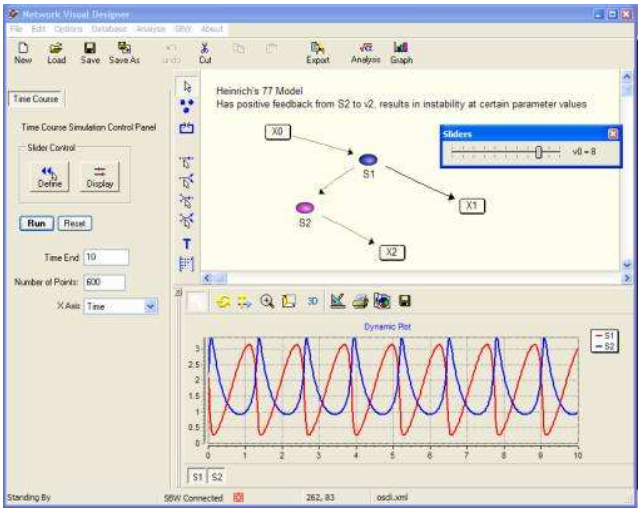


Abbildung 1.11: JDesigner Pathway-Editor/Analyse JDesigner ist ein Windows-Programm, das vornehmlich zur interaktiven Darstellung von metabolischen Pfaden entwickelt wurde. Über das SBML-Dateiformat (Hucka *et al.*, 2002b) können aber auch einige Analyse-Methoden (z.B. Pfeiffer *et al.*, 1999) eingebunden werden.

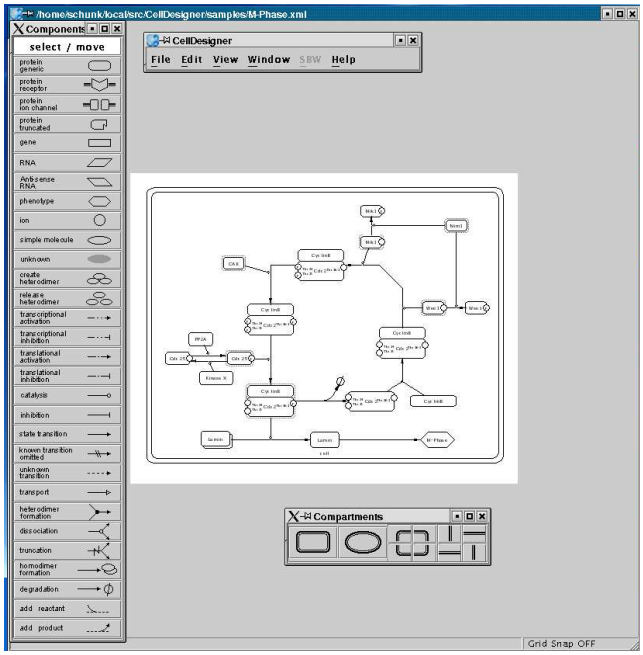


Abbildung 1.12: CellDesigner, der Nachfolger von JDesigner CellDesigner ist eine verbesserte Reimplementation von JDesigner.

experimentelle Daten abgebildet und z.B. Korrelationsanalysen ausgeführt werden. Die experimentellen Daten werden dabei im Graphen direkt angezeigt (siehe Abbildung 1.13 auf der nächsten Seite). Es besteht auch die Möglichkeit, das Netzwerk manuell zu formatieren und mittels „Gravisto“ ein Layout zu berechnen (Passau, University, 2006). Obwohl VANTED historisch als Nachfolger von *BioPath* betrachtet werden kann, verfügt es über keine eigenen metabolischen Daten oder Schnittstellen zu anderen Datenbanken.

1.2.16 Cytoscape

Das ebenfalls zur Darstellung und Analyse von metabolischen Netzwerken geschaffene Programm Cytoscape (Shannon *et al.*, 2003) ist ähnlich zu VANTED. Die experimentellen Daten werden aber nicht direkt im Graphen angezeigt, sondern stehen über diverse Textfelder zur Verfügung. Dieses Programm liest zusätzlich auch Dateien im SMBL-Format und kann die Annotation für Gene aus der KEGG und Gene Ontologie (Consortium, 2000) importieren. Die Layoutmethoden stammen alle aus der Graph-Bibliothek von Java. Die Lesbarkeit der ausgerichteten Graphen ist aber nicht sehr gut. Die Möglichkeiten Netzwerkelemente zu formatieren, sind eingeschränkt, dafür können aber sehr große Graphen (>10.000 Knoten) dargestellt werden (siehe Abbildung 1.14 auf der nächsten Seite). Über eine Pluginschnittstelle kann die Funktionalität von Cytoscape erweitert werden.

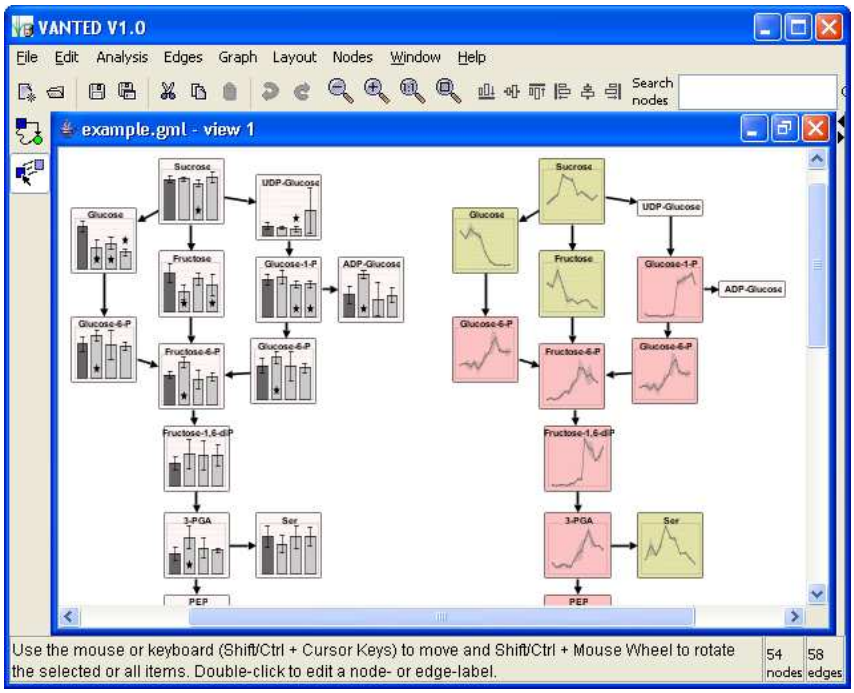


Abbildung 1.13: Visualisierung mit VANTED Der Screenshot zeigt das Hauptfenster von VANTED. Man kann sehr gut erkennen, dass experimentelle Daten direkt in die Knoten gerendert werden können.

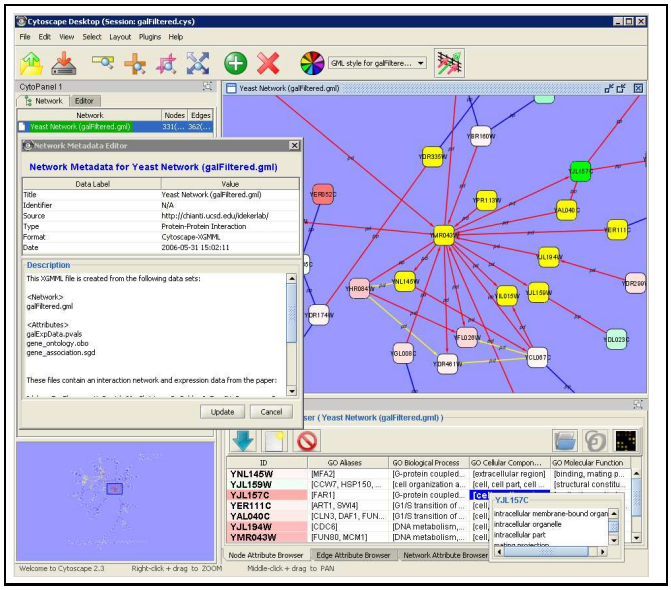


Abbildung 1.14: Analysen mit Cytoscape Die Abbildung zeigt den Screenshot einer Beispielsitzung. Cytoscape ist in der Lage, sehr große Graphen darzustellen. Diese Graphen können analysiert und mit experimentellen Daten verknüpft werden.

Tabelle 1.1: Eigenschaften von Programmen zur Darstellung und Analyse von metabolischen Pfaden
Zur Erläuterung der Kriterien siehe Abschnitte 1.1.1 bis 1.1.4. Nicht vorhanden: —, sehr gut: (++), gut: (+), neutral: (±), schlecht: (−), sehr schlecht: (−−).

	GSCope	Pathway Tools	PathViewer	JDesigner	*PathFinder*	*BioPath*	PATIKA	DMD	KEGG	BioJAKE	ExPASy	Java-Editor	GeneNet	UM-BBD	VANTED	Cytoscape
Publiziert	2002	2002	2002	2002	2002	2001	2002	2001	2000	1999	1999	2002	1999	1998	2006	2003
Benutzerschnittstelle																
WWW-Interface	(±)	(±)	—	—	(±)	(+)	—	(±)	(++)	—	(±)	—	(±)	(++)	—	—
Editor	—	—	(−)	(±)	(−−)	—	(+)	—	—	(±)	—	(±)	(−)	—	(+)	(+)
Automatisches Layout																
Lesbarkeit	—	(+)	(−−)	—	(−)	(+)	(±)	(−−)	—	—	—	—	—	(±)	(+)	(−)
Nebenmetabolite	—	(++)	(−)	—	—	(+)	—	—	—	—	—	—	—	—	—	—
Mentale Karte	—	(±)	—	—	—	(±)	(−)	—	—	—	—	—	—	(+)	—	—
Gruppieren	—	—	—	—	—	—	—	—	—	(±)	—	—	—	—	—	(−)
Fokussieren	—	(+)	—	—	—	(−)	—	—	—	—	—	—	—	—	—	(+)
Runtime-Layout	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Manuelle Formatierung																
Geometrische Form	—	—	—	(+)	—	—	(+)	(−)	—	(±)	—	(+)	(−)	—	—	—
Strichart/-stärke	—	—	—	(+)	—	—	(+)	(−)	—	(±)	—	(+)	(−)	—	(±)	(±)
Beschriftung	—	—	—	(+)	—	—	(+)	(−)	—	(±)	—	(+)	(−)	—	(+)	(+)
Reaktionswege	—	—	—	(+)	—	—	(+)	(−)	—	(±)	—	(+)	(−)	—	(+)	(+)
Graphenlayout	—	—	—	(+)	—	—	(+)	(−)	—	(±)	—	(+)	(−)	—	(±)	(±)
Methoden																
Subgraphen	—	(+)	(−)	(+)	(−)	—	—	(±)	—	—	—	—	(±)	—	—	—
Pfade	—	(±)	—	(+)	(+)	(+)	—	(±)	—	—	—	—	—	(+)	—	—
Simulation/Analyse	—	—	—	(+)	—	—	—	(±)	—	—	—	—	—	—	(++)	(++)
Vergleiche	(±)	(±)	—	(+)	(−)	(±)	(+)	(±)	(+)	(−)	—	—	—	(+)	(+)	—
Datenverifizierung	—	—	—	(+)	—	—	—	—	—	—	—	—	—	—	—	—
Moleküldarstellung	—	(++)	—	—	—	(±)	—	—	(+)	—	(+)	—	(±)	(+)	(±)	—
Verknüpfungen	(−)	(++)	(±)	—	(−−)	(±)	(+)	(±)	(+)	(−)	(++)	—	(±)	(++)	(±)	(±)

1.3 Metabolische Netzwerke

Eine einfache metabolische *Reaktion* beschreibt die biochemische Umsetzung von Substraten zu Produkten. Die meisten Reaktionen in einer Zelle werden von Enzymen katalysiert.

$$ATP + \text{D-Glucose} \xrightleftharpoons{Glucose-Phosphotransferase} ADP + \text{D-Glucose-6-phosphat}$$

Solche Reaktionen kann man schematisch wie folgt notieren:

$$A + B \xrightleftharpoons{Enzym} C + D$$

Eine Folge von Reaktionen, die von einem Ausgangsstoff zu einem bestimmten Endprodukt führen, nennt man einen *Pfad*.

$$A + B \xrightleftharpoons{E1} C + D \xrightleftharpoons{E2} E + F \xrightleftharpoons{E3} G + H$$

Häufig kommt es aber auch vor, dass Enzyme mehr als eine Reaktion katalysieren und dass Metabolite an vielen Reaktionen teilnehmen. All diese Reaktionen bilden also ein Netzwerk und können nicht isoliert betrachtet werden.

$$A + B \xrightleftharpoons{E1} C + D \xrightleftharpoons{E2} E + F \xrightleftharpoons{E3} G + H$$
$$C + I \xrightleftharpoons{E2} E + J \xrightleftharpoons{E4} G + K$$
$$A + L \xrightleftharpoons{E5} C + M$$

Um in diesem sehr unübersichtlichen Netzwerk (vergleiche Michal, 1999) dennoch den Überblick zu behalten, werden die Reaktionen, die abgrenzbare Biosyntheseprozesse beschreiben, zu *Pathways* zusammengefasst. Viele der so abgegrenzten Subnetzwerke tragen in der Biochemie etablierte Namen (z.B. „Gluconeogenese“, siehe Lehninger *et al.*, 1994).

1.3.1 Hochvernetzte Metabolite

Biochemische Reaktionsnetzwerke unterscheiden sich von zufälligen Netzwerken. Beispielsweise gibt es Metabolite, die an besonders vielen Reaktionen teilnehmen (z.B. ATP: 2121, Pyruvat: 661, Glucose: 519). Im gezeichneten Reaktionsnetzwerk fallen solche Metabolite durch zahlreiche ein- und ausgehende Kanten auf. Dieser hohe Knotengrad führt zu sehr langen Kanten und vielen Kantenüberkreuzungen (siehe Abbildung 1.2 auf Seite 8).
Ohne eine besondere Behandlung können Metabolite wie *Wasser* (> 12.000 Reaktionen) gar nicht in den Graphen eingezeichnet werden, da dieses zu einer völlig unlesbaren Zeichnung führen würde. Zudem würde der hohe Knotengrad solche *Nebenmetabolite* völlig überbetonen. Diese *Nebenmetabolite* sind zwar für eine stöchiometrisch korrekte Darstellung notwendig, aber sie sind im Zellplasma oder den entsprechenden Organellen in ausreichend hoher Konzentration und gleichmäßig verteilt verfügbar. Es gibt in einer Zelle also ein Reservoir oder *Pool* von *Nebenmetaboliten*.

1.3.2 Grafische Darstellung

Wie in den vorausgegangenen Abschnitten gezeigt (ab 1.2 auf Seite 6), werden die Graphen von Biosyntheseprozessen unterschiedlich gezeichnet. Prinzipiell lassen sich folgende Darstellungsformen feststellen.

Metabolitgraph

In einem metabolitzentrierten Graphen werden die Produkte und Substrate des Netzwerkes als Knoten gezeichnet. Die Kanten zwischen diesen Knoten stellen die Reaktionspfeile dar und können gerichtet sein. Häufig sind den Kanten Attribute bzw. Gewichte zugeordnet. Diese Attribute können zum Beispiel Enzyme bezeichnen oder Reaktionskinetiken wiedergeben.

Enzymgraph

Häufig werden zur Darstellung regulatorischer Prozesse die Enzyme als Knoten gezeichnet. Die Kanten zwischen den Knoten können dann die Art der Regulation ausdrücken. In einem solchen Netzwerk werden die Metabolite dann nicht oder nur rudimentär eingezeichnet.

Enzym- und Metabolitgraph

Der Enzym- und Metabolitgraph besteht aus zwei unterschiedlichen Arten von Knoten. Die eine Knotenart symbolisiert dabei dann die Metabolite, die andere die Enzyme. Eine Kante liegt dann immer zwischen einem Enzym- und einem Metabolitknoten. Eine Kante stellt so also nur eine Teilreaktion dar und bezeichnet entweder die „rechte“ oder die „linke“ Seite einer Reaktionsgleichung. Diese Graphen geben die Vernetzung von Metaboliten und Enzymen direkt wieder, erhöhen aber die Anzahl der Kanten.
In *Cupe* wird ein metabolisches Netzwerk standardmäßig als Enzym- und Metabolitgraph gezeichnet. Es ist in *Cupe* aber auch möglich, den Graphen enzym- oder metabolitzentriert zu zeichnen.

Reaktionsgraph

Für Simulationen des Metabolismus einer Zelle wird häufig die Darstellung als Reaktionsgraph gewählt. Hierbei stellen die Knoten die Einzelreaktion dar und die Verknüpfung über die Kanten das Simulationsmodell.

1.3.3 Mathematische Abbildung

Das gezeichnete Netzwerk in *Cupe* entspricht mathematisch einem Graph G.

$$G = (V, E) \text{ (V Menge der Knoten, E Menge der Kanten)}$$

Viele Kanten in diesem Graphen sind gerichtet. Anschaulich bedeutet das, dass eine Kante immer von einem Knoten v zu einem anderem Knoten v‘ verläuft und dass ein so gebildetes Knotenpaar $(v, v‘)$ ungleich $(v‘, v)$ ist. Die Knoten V im Graph G lassen sich in zwei disjunkte Teilmengen aufteilen: einmal die Menge der Enzymknoten und einmal die Menge der Metabolitknoten. Da innerhalb dieser beiden Teilmengen keine Kanten e verlaufen, sagt man, dass der Graph bipartit ist. In *Cupe* wird der Graph G nur benutzt, um das Layout des metabolischen Netzwerkes zu berechnen. Dass die meisten Kanten gerichtet sind und der Graph bipartit ist, findet keine weitere Beachtung.
Eine andere Art der Darstellung wird in *Cupe* über zweidimensionale Matrizen bereitgestellt. Diese Matrizen sind extra für *Cupe* entwickelt worden und werden im Kapitel „Ergebnisse“ (3.4.1 auf Seite 63) vorgestellt.

2 Material und Methoden

Die Angaben in den nächsten Abschnitten ermöglichen es, eine voll funktionsfähige *Cupe*-Entwicklungsumgebung auf einem anderen Computersystem einzurichten. Besondere Sorgfalt bei der Einrichtung der Entwicklungsumgebung wird durch die Verwendung der *LEDA*-Bibliothek in der Version 4.5 nötig (siehe Abschnitt 2.5.3 auf Seite 38). Hierdurch sind zahlreiche Einschränkungen zu beachten, die erst mit dem Wechsel der *AGD*-Bibliothek (siehe 2.5.2) zur *OGDF*-Bibliothek (siehe 3.3) aufgehoben werden können.

2.1 Entwicklungsumgebung

Wenngleich die gesamte Entwicklung von *Cupe* in einer Linuxumgebung stattgefunden hat, wurde während der Entwicklung dennoch darauf geachtet, dass die benutzten Programme und Bibliotheken keine prinzipielle Abhängigkeit zu Linux hervorrufen. Spätestens mit der Ablösung von *AGD* (siehe 2.5.2 auf Seite 37) durch *OGDF* (siehe 3.3 auf Seite 55) sollte eine Portierung zu *Microsoft Windows* durchführbar sein.

2.1.1 Hardware

Als Entwicklungsplattformen dienten Personal Computer mit *Intel*-Prozessoren (Tabelle 2.1). Als Betriebssystem wurde *Linux* in den Distributionen von Debian (3.0 und 3.1) eingesetzt. Die Versionsnummern der *Linuxkernel* waren 2.6.8 und 2.6.15.

Tabelle 2.1: Benutzte Computersysteme Informationen zur Hardware der benutzten Computersysteme. Beim Compilieren und Testen der Programme ist zu bedenken, dass der Speicherbedarf mit der Größe der Datenbank und der Anzahl der Kommandos im *Commandmanagement* steigt. Ein Computer mit wenig Arbeitsspeicher (< 512 MB) kann daher schnell überlastet werden. Gleichwohl können mit einem solchen Rechner die Programmquellen übersetzt und das Programm getestet werden.

Computer	CPU	RAM
Desktop-Systeme	2400 MHz	1024 MB
Notebook	1000 MHz	640 MB
Notebook	1600 MHz	1024 MB

2.1.2 Compiler und Hilfsprogramme

Cupe wird mit den C++-Compiler der *GNU Compiler Collection (GCC)* (Versionen 3.1 bis 3.3) übersetzt. Andere Compiler können nicht benutzt werden, solange die *AGD*-Bibliothek zum Zeichnen von Graphen verwendet wird.
Die Kompilierung wird über das Programm *GNU-Make* gesteuert und ist unter der Zuhilfenahme der *GNU-Autotools* weitgehend automatisiert worden. Für die Synchronisation der verteilten Entwicklung von *Cupe* wird das *Concurrent Versions System (CVS)* genutzt.

2.2 Programme

Für die Entwicklung von *Cupe* wurden eine Vielzahl von Programmen genutzt. Diese Programme müssen nicht vorhanden sein, um die Programmquellen von *Cupe* weiter zu entwickeln. Sie stellen vielmehr eine Auswahl von Werkzeugen dar, die in Analogie zu der Methodik in einem Labor die Programmierung von *Cupe* bestmöglich unterstützen. Die Programmentwicklung spiegelt aber auch die Auswahl dieser Programme wieder. Der Wechsel zu anderen Programmen ist von daher auch stets mit der Anpassung der Methodik der Programmentwicklung von *Cupe* verbunden. Beispielsweise ist die Dokumentation des Programmcodes an die Bedürfnisse von *Doxygen* angepasst.

2.2.1 Editor und Textformatierung

Alle Textdateien (Programmquellen, Doktorarbeit usw.) wurden mit dem Editor *Nedit* (Version 5.5) erstellt. Dieser Editor unterstützt das *X Windowsystem* (X.Org Foundation), Syntaxhervorhebung und vieles mehr. Gleichzeitig bietet er eine einfache und mächtige Makrosprache, um eigene Erweiterungen zu schreiben. Mittels solcher Makros konnte beispielsweise die Programmentwicklung erheblich erleichtert und beschleunigt werden.

2.2.2 Doxygen

Viele von den in dieser Dissertation implementierten neuen C++-Klassen können auch in anderen Programmen genutzt werden (z.B. die Klassen *cubic::sparse_matrix* und *cubic::properties*). Dieses kann und wird jedoch nur geschehen, wenn der dazugehörige Quellcode sinnvoll dokumentiert ist. Für *Cupe* wurde eine solche Dokumentation mit der Unterstützung von *Doxygen* (Version 1.5.1) erstellt. Dieses Dokumentationssystem gestattet zum einen, die Dokumentation direkt innerhalb der Programmquellen zu schreiben und bietet zum anderen eine direkte Unterstützung der Sprachmittel von C++ an (z.B. die graphische Darstellung der Vererbung). Die so erstellte Dokumentation kann sowohl als HTML-Seite im Internet zur Verfügung gestellt als auch mittels LaTeX formatiert auf Papier gedruckt werden.

2.2.3 GNU Binutils

Bei den *GNU Binutils* handelt es sich um ein Sammlung von Programmen, um die von Compiler generierten Objektdateien zu untersuchen und zu Bibliotheken bzw. ausführbaren Programmen zusammenzustellen.

nm

Eines der Programme aus dieser Sammlung ist z.B. *nm*. Dieses Programm kann die Symbole innerhalb von Objektdateien und Bibliotheken anzeigen und analysieren. Dieses ist insbesondere bei der Entwicklung von Plugins hilfreich, da hier der Linker nicht prüfen kann, ob alle Symbole und Sprungadressen eines Plugins aufgelöst werden. Dadurch können unvollständig definierte Deklarationen erst zur Laufzeit von *Cupe*, wenn alle notwendigen Bibliotheken geladen sind, erkannt werden. Mittels *nm* ist es dann möglich, solche Fehler zu lokalisieren und zu beheben.

gprof

In *Cupe* werden sehr viele Positionsberechnungen für graphische Elemente und Mengenoperationen auf sehr großen Datenstrukturen durchgeführt. Es ist daher vorgekommen, dass sich ineffiziente Programmroutinen in einer sehr langsamen Programmdurchführung ausdrückten. Um solche Schwachstellen in der Programmierung aufzudecken und zu beheben, können sogenannte Profiler eingesetzt werden. Diese Profiler dokumentieren während des Programmablaufs, wieviel Zeit die einzelnen Programmroutinen beanspruchen. Bei der Entwicklung von *Cupe* ist *gprof* eingesetzt worden, um z.B. die Zugriffe auf die *cubic::sparse_matrix* zu optimieren (siehe 3.4.1 auf Seite 63).

2.2.4 GNU gdb und DDD

Die Entwicklung von kompletten Anwendungen erfolgt nur in den seltensten Ausnahmefällen durchgehend fehlerfrei. Insbesondere die intensive Nutzung von Zeigerdatentypen, wie sie durch die Verwendung der *Qt*-Bibliothek nötig werden, führt häufig zu Programmabbrüchen. Leider treten gerade die Abbrüche, die durch Speicherzugriffsfehler ausgelöst werden, häufig nicht systematisch auf. In solchen Fällen ist ein genaues Protokoll der jeweiligen Programmzustände nötig. Hierzu wurde der Debugger *GDB* (Version 6.4.90) eingesetzt. Dieser Debugger kann sich jederzeit an ein laufendes und entsprechend vorbereitetes Programm binden und anzeigen, welchen Status es gegenwärtig einnimmt oder – was ebenfalls von beträchtlichem Nutzen ist – womit ein Programm beschäftigt war, bevor dieses einen undefinierten Zustand einnahm und seine Ausführung abbrach. Gerade für die Fehlersuche wurde der *GDB* in Verbindung mit der grafischen Oberfläche des *Data Display Debugger (DDD)* (Version 3.3.11) eingesetzt.

2.2.5 Gimp

Das *GNU Image Manipulation Program* ist ein professionelles Bildbearbeitungsprogramm, das für viele Betriebssysteme zur Verfügung steht. Alle Icons, Grafiken und Splashscreens von *Cupe* sind mit diesem Programm erstellt worden. Wenn für *Cupe* oder Plugins von *Cupe* neue Icons gebraucht werden, können diese mit *Gimp* und bereits angefertigten Vorlagen leicht und schnell angefertigt werden.

2.3 Programmiersprachen

Prinzipiell lassen sich Programmiersprachen in *interpretierte* Sprachen und *compilierte* Sprachen unterteilen. Interpretierte Sprachen wie z.B. Perl, Ruby oder Shellscripte bieten eine stärkere Abstraktion von systemimmanenten Aspekten (beispielsweise Typisierung oder Deklarationen) und ermöglichen damit dem Entwickler die Konzentration auf die eigentliche Problemstellung. Dafür müssen sie aber von „Dolmetschern“ gelesen und unter ihrer Kontrolle ausgeführt (interpretiert) werden. Interpretierte Sprachen sind dadurch immer langsamer, als compilierte Sprachen. In die Art und Weise der Ausführung und die Systemzugriffe durch den Interpreter hat der Programmierer kaum Einflussmöglichkeiten. Compilierte Sprachen wie C oder C++ haben diese Einschränkungen nicht. Dafür muss der Programmierer die Aufgaben des Interpreters sozusagen mit übernehmen, was die Entwicklung großer Programme sehr komplex macht. Die Komplexität großer Programme lässt sich aber durch ein stringentes Design und die Nutzung etablierter Entwurfsmuster kontrollieren.

2.3.1 Die Programmiersprache C++

Das Programm *Cupe* wurde vollständig in C++ (Stroustrup, 2000) geschrieben. Diese Sprache unterstützt objektorientierte Datenabstraktionen direkt. So kann man eigene Datentypen und die auf ihnen erlaubten Operationen, unabhängig von ihrer konkreten Implementation, auf einer hohen Abstraktionsebene deklarieren und über einheitliche Schnittstellen zur Verfügung stellen. Diese Arbeitstechnik wird durch die Sprachmittel von C++ (Klassen, Vererbung, Polymorphie, Templates, usw.) entscheidend erleichtert. C++ bietet zudem auch Methoden zur Erhaltung der Datenkonsistenz und eine erleichterte Verwaltung des Speichers. Insbesondere dann, wenn auf ihn über die Container der Standard Template Library (siehe 2.1.2 auf Seite 24) zugegriffen wird.

2.3.2 Interpretierte Sprachen

Die Schnittstelle zwischen Hardware und Programmen auf der einen Seite und Anwender auf der anderen Seite bilden Kommandointerpreter, die sogenannten Shell's. Auf den in Abschnitt 2.1 auf Seite 23 vorgestellten Computersystemen wurde die

Bourne-Again Shell (bash) eingesetzt. Diese Shell bietet eine umfangreiche Skriptsprache, mittels der man Prozessabläufe konfigurieren und automatisieren lassen kann. Diese Skriptsprache wurde eingesetzt, um die Binärdistribution von *Cupe*, mit Hilfe von Umgebungsvariablen, an die Zielplattform anzupassen.

2.4 Programmiermethodik

Im Umfeld der akademischen Softwareentwicklung müssen eine Reihe von Besonderheiten berücksichtigt werden. Insbesondere der häufige Wechsel der Entwickler und ihre unterschiedlichen Vorkenntnisse machen eine einheitliche Programmiermethodik nötig. Die in *Cupe* praktizierte Methodik fokussiert besonders die Unabhängigkeit der Programmquellen bzw. -abläufe vom jeweiligen Entwickler und erstreckt sich über nahezu alle Aspekte der Programmentwicklung.

2.4.1 Ausdrucksweise im Programmcode

In professionell entwickelten Programmen existiert ein einheitlicher Programmierstil (Codingstyle); dieser wird im Allgemeinen in Richtlinien festgehalten und ist für alle Entwickler bindend. Eine derartige Richtlinie lässt sich in einer Universität jedoch nicht durchsetzen. Zum einem, weil die einzig beste Richtlinie nicht existiert und zum anderen, weil keine durchgängige und strenge Quellcodebegutachtung und -bewertung durchgeführt werden kann. So werden sich die Stile mit jedem Entwickler ändern. Gleichwohl ist die Einhaltung eines Stils von großem Vorteil, wenn sich fremde Entwickler in Programmquellen einarbeiten müssen. In *Cupe* wurde daher stets darauf geachtet, dass innerhalb eines Moduls immer der gleiche Codingstyle eingehalten wurde. Der Codingstyle von *Cupe* ist im Programmbeispiel 2.1 auf der nächsten Seite beispielhaft demonstriert. Programmteile, die während Praktika entstanden sind und der Sourcecode von Plugins, können aber von den Konventionen in *Cupe* abweichen. Im Allgemeinen produziert man einen gut lesbaren Programmcode, wenn man sich folgenden Merksatz regelmäßig vergegenwärtigt: „*Programme werden für Menschen geschrieben!*“ Dem Compiler wäre es völlig gleichgültig wie gut verständlich ein Programm geschrieben ist, er würde mit Nullen und Einsen auskommen.

2.4.2 Intrinsische Dokumentation

Intrinsisch bedeutet: „Von innen her kommend“. Intrinsische Dokumentation meint daher, dass die Dokumentation aus dem Algorithmus selbst heraus kommt und zugleich die Funktionalität des Algorithmus definiert. Es ist deshalb keine äußere *zusätzliche* Dokumentation nötig. Der überragende Vorteil dieser Technik ist, dass die Dokumentation nicht asynchron zur Funktion werden kann. Dieses Ziel lässt sich durch die gute Namenswahl für Operanden (Variablen) und den Relationen (Funktionen) zwischen ihnen erreichen. Zusätzlich ist es noch nötig, auf abstrakte Parameter (*magic numbers*) zu verzichten und diese stattdessen mit der Hilfe von

Quellcode 2.1: Ausdrucksweise im Programmcode Dieses Listing zeigt einige der in *Cupe* benutzten Konventionen zum Programmierstil. Alle Teile eines Bezeichners werden mit Unterstrichen voneinander getrennt. Klassen und Strukturnamen beginnen mit Großbuchstaben, Membervariablen werden klein geschrieben und enden mit einem Unterstrich. Die Werte von Aufzählungstypen werden durchgängig groß geschrieben (Zeile 4). Wo es sinnvoll ist, wird Leerraum eingefügt, um die Lesbarkeit zu erhöhen.

```
1  struct Molecule_Info
2  {
3      unsigned int                        index_;
4      enum Type { ENZYME, COMPOUND }      type_;
5
6      Molecule_Info() : index(0), type( COMPOUND ) {}
7
8      Molecule_Info( unsigned int i, Type t )
9      :   index_( i ), type_( t )
10     {}
11
12     Molecule_Info( const Molecule_Info& m )
13     : index( m.index_ ), type( m.type_ ) {}
14
15     Molecule_Info&
16     operator=( const Molecule_Info& other )
17     {
18         if ( this == &other ) return *this;
19
20         index_ = other.index_;
21         type_  = other.type_;
22
23         return *this;
24     }
25 };
```

Aufzählungstypen vollständig auszuformulieren (siehe Programmbeispiel 2.2 auf der nächsten Seite).

2.4.3 Automatische Dokumentation

Damit Neueinsteiger in ein komplexes Projekt einsteigen können, ist eine Dokumentation beziehungsweise ein Design-Dokument unabdingbar. Vor allem bei objektorientierten Programmen wie *Cupe* sind die zugrunde gelegten Datenmodelle nur schwer aus dem Quellcode zu extrahieren. Außerdem steht man vor dem Problem, dass neue Entwickler häufig neue Programmteile schreiben, beziehungsweise die bestehende Klassenhierarchie ändern, ohne dieses zu dokumentieren (siehe 2.4.2). Um diese Probleme zu umschiffen, ist eine automatisch erstellte Dokumentation notwendig. In *Cupe* werden Kommentare deshalb in einem speziellen Format geschrieben und mit *Doxygen* (2.2.2) zu einer umfangreichen Dokumentation zusammengestellt (siehe Programmbeispiel 2.3 auf Seite 31).

2.4.4 Sourcecodeaufteilung

Für *Cupe* werden die Deklarationen in Dateien mit der Endung *.hpp und die Definitionen in Dateien mit der Endung *.cpp-Dateien geschrieben. Damit man erkennt, welche Dateien zu welchen Modulen gehören, ist der Quellcode in entsprechend benannte Verzeichnisse und Unterverzeichnisse aufgeteilt (vergleiche Quellcodeverzeichnis 2.4 auf Seite 32).

2.4.5 Sprachmittel

Cupe nutzt nahezu alle Sprachmittel von C++ und „Template Meta Programming“ (Vandervoorde & M., 2003) in einer professionellen und modernen Art und Weise. Die Entwicklung von *Cupe* zeichnet sich dabei aber stets durch die enge Anlehnung an sehr gut dokumentierte Techniken und etablierte Methoden aus (siehe Meyers, 2005, 1997; Sutter, 2000; Stroustrup, 2000) und verzichtet weitgehend auf eigene „Neuschöpfungen“.

Klassen, Vererbung, Polymorphie

Das Wesen der objektorientierten Programmierung liegt darin, Daten, Eigenschaften und Methoden in *Klassen* zusammenzufassen und gemeinsam in einem Objekt anzusprechen. Solche Klassen können durch *Vererbung* spezialisiert und im Verhalten angepasst werden (*Polymorphie*). In *Cupe* ist jedes Element auf einer Zeichenfläche (3.7.3) ein *Cupe_Item*. Jede Spezialisierung (*Cupe_Node, Cupe_Enzyme, Cupe_Edge* usw.) wird von dieser Basisklasse abgeleitet (vergleiche Quellcode 2.5 auf Seite 33). Eine besondere Variante einer Klasse stellen die so genannten *Funktionsobjekte* dar. *Funktionsobjekte* ermöglichen es, eigene Daten über *Iteratoren* mit eigenen Algorithmen zu prozessieren. Auf diese Weise sind Daten und Algorithmen getrennt und

Quellcode 2.2: Intrinsische Dokumentation Das Ziel der intrinsischen Dokumentation ist es, die Dokumentation aus dem Programmcode selbst abzuleiten. Durch die gute Wahl des Funktionsnamens und die Verwendung von Aufzählungstypen ist der Funktionsaufruf in Zeile 3 selbsterklärend. Die aufgerufene Funktion (ab Zeile 7) ist ebenfalls ohne weitere Dokumentation leicht zu verstehen. Im aufgelisteten Beispiel wird so die Ausführung einer sehr aufwendigen Funktion aus der Qt-Bibliothek kontrolliert (Zeile 28).

```
1  void foo()
2  {
3    board->update( Cupe_Board::SUPPRESS );
4  }
5
6  void
7  Cupe_Board::update( Cupe_Board::Update_Attitude attitude )
8  {
9      switch ( attitude )
10     {
11         case SUPPRESS:
12             repress_update_ = true;
13             emit repress_updates();
14             break;
15
16         case REFRESH:
17             repress_update_ = false;
18             emit accept_updates();
19             break;
20     }
21 }
22
23 void
24 Cupe_Board::redraw()
25 {
26     if ( ! repress_update_ )
27     {
28         QCanvas::update();
29     }
30 }
```

Quellcode 2.3: Automatische Dokumentation mit Doxygen Das Programm *Doxygen* kann Programmquellen automatisch auswerten und eine externe Dokumentation erstellen (z.B. HTML). Durch die Verwendung von Schlüsselwörtern (@class, @brief usw.) ist es möglich, den Umfang der generierten Dokumentation zu erweitern.

```
1  /*! @class  sparse_matrix ./sparse_matrix.hpp sparse_matrix.hpp
2   *  @brief A matrix which dosen't store the sparse value
3   *  @author schunk
4   *  @date Wed Oct  1 14:03:03 CEST 2003
5   *
6   *  This template sparse matrix class is consistent with the
7   *  C++ANSI-Standard. This class works much like a two-
8   *  dimensional matrix (e.g. int[5][5]), but it won't store
9   *  these values which are defined to be the 'sparse value'
10  *  (=T()). The sparse_matrix builds up on sgi's hash_map.
11  *  Most public members of sgi's hash_map are available.
12  *  @see http://www.sgi.com/tech/stl/hash_map.html
13  *
14  *  @see @ref SGI (embedded webpage)
15  *  \warning the type of I must be an integer type
16  *
17  */
18 template<typename T, typename I = unsigned long >
19 class sparse_matrix
20 :   private __gnu_cxx::hash_map<std::pair<I,I>, T, hash_func>
21 {
22   ...
23 public:
24   /*! @brief Constructor
25    *  @param sv sparse value
26    *  @param n buckets at start up
27    *
28    *  The default sparse value is equal to mapped();
29    *  The default value of n is 0;
30    *
31    */
32   sparse_matrix( const mapped_type& sv =mapped(), size_type n=0 )
33   :   hash_map_type(n),
34       sparse_value_(sv),
35       row_layout_( matrix_layout::row_column ),
36       column_layout_( matrix_layout::column_row )
37   {}
38   ...
39 };
```

Quellcode 2.4: Quellcodeverzeichnis Die frisch aus dem *CVS-Repository* ausgelesenen Programmquellen von *Cupe* umfassen insgesamt 1340 Dateien. Für eine bessere Übersicht wurden die Quellen in unterschiedliche Verzeichnisse – jeweils mit dem Präfix *Cupe* – aufgeteilt.

```
1  @cupe> ls
2
3  aclocal.m4                  Cupe_Network
4  AUTHORS                     Cupe_Observer
5  autom4te.cache              Cupe_Parser
6  ChangeLog                   Cupe_Plugin
7  config.h.in                 Cupe_Properties
8  configure.bin_dist.in       Cupe_Reaction
9  configure.bin_install.in    Cupe_Splash
10 configure.in                Cupe_Tools
11 configure.src_install.in    CVS
12 COPYING                     doc
13 Cubic_Matrix                Doxyfile
14 Cupe                        examples
15 Cupe_Action                 include
16 Cupe_Analyzer               INSTALL
17 Cupe_Animation              install-sh
18 Cupe_Board                  lib
19 Cupe_Canvas_Composite       Makefile.bin_dist.in
20 Cupe_Drawing_Area           Makefile.in
21 Cupe_Editor                 Makefile.src_install.in
22 Cupe_File_IO                README
23 Cupe_Icons                  run_cupe.bin_dist.sh.in
24 Cupe_Info                   Cupe_Layout
25 Cupe_Item                   run_cupe.sh
26 Cupe_Ligand                 run_cupe.sh.in
27 Cupe_Memory
```

Quellcode 2.5: Cupe_Item *Cupe_Item* ist die Basisklasse für alle Objekte, die auf einer Zeichenfläche gezeichnet werden und mit dem Mauszeiger interagieren können.

```
1  class Cupe_Item
2    : public QObject, boost::noncopyable
3
4  class Cupe_Border
5    : public Cupe_Item
6
7  class Cupe_Node
8    : public Cupe_Item, public Cupe_Network_Subject<Cupe_Node>
9
10 class Cupe_Molecule
11   : public Cupe_Node
12
13 class Cupe_Metabolit
14   : public Cupe_Molecule
15
16 class Cupe_Edge
17   : public Cupe_Item , public Cupe_Network_Subject<Cupe_Edge>
18
19 ...
```

können bei Bedarf beliebig über Iteratoren miteinander verbunden werden. Ein Beispiel für diese Technik ist im Programmbeispiel Observer (siehe 3.2 auf Seite 81) in Zeile 5 zu sehen.

Templates

Bei Templates handelt es sich um Schablonen von Algorithmen, die zur Kompilierzeit für einen bestimmten Datentyp spezialisiert werden. Sie stellen sozusagen das statische Gegenstück zur dynamischen Bestimmung der Vererbungshierarchie während der Laufzeit dar. Ein Vorteil von Templates liegt darin, dass die von ihnen durchgeführten Algorithmen durch den Compiler optimiert werden können. Templates gehören unter den Designmustern zu den Verhaltensmustern (Gamma *et al.*, 1996) und werden in *Cupe* sehr intensiv genutzt (siehe z.B. Programmbeispiele 2.5, 2.3 und 3.7).
Templates können von einem Compiler zu einem einzigen Ausdruck ausgewertet werden. Das führt – gerade für komplexe Templates – zu einer verlängerten Kompilierzeit, kann aber die Ausführungsgeschwindigkeit bestimmter Algorithmen sehr beschleunigen. Eine solche Technik stellt eine Spezialisierung der allgemeinen Templates zum so genannten *Template Meta Programming* dar. Die Methodik des *Template Meta Programming* wird für alle selbst definierten Cupe-Parser genutzt (Verzeichnis `Cupe_Parser` siehe 2.4 auf der vorherigen Seite).

2.4.6 Designmuster

Objektorientierte Designmuster sind wiederverwertbare und auf viele Programmiersprachen übertragbare Konzepte, um eine genau definierte Problemstellung zu lösen. Dabei ist das benutzte Muster nahezu unabhängig von der Art seiner Programmierung. Ein Programmierer, der weiß, welche Designmuster zur Implementierung eines Moduls genutzt wurden, kann eigene Entwicklungen beisteuern, ohne die Implementation im Detail zu kennen. Er hat sozusagen das „Große und Ganze“ im Überblick und kann sich problemlos orientieren. Übertragen auf die Labortechnik, sind Designmuster beispielsweise äquivalent zu den Abläufen während einer Proteinaufreinigung. In *Cupe* werden ausschließlich bekannte und allgemein etablierte Entwurfsmuster genutzt bzw. kombiniert und erweitert. Wenn ein Entwickler nicht mit den Entwurfsmustern innerhalb der von ihm zu bearbeitenden Programmmodule vertraut ist, kann er auf die Literatur zurückgreifen und die Prinzipien nachlesen (siehe dazu Gamma *et al.*, 1996; Alexandrescu, 2001).

Erzeugermuster

Um über eine gleichartige Schnittstelle unterschiedliche Elemente des metabolischen Netzwerks zu zeichnen, werden die Erzeugermuster *Abstrakte Fabriken* und *Fabrikmethoden* genutzt. Dieser Ansatz ermöglicht es, die Elemente der Graphen zu verändern und zu erweitern, ohne dass die zugrunde liegenden Programmabläufe geändert werden müssen. Auf diese Weise können neue Netzwerkelemente sogar dynamisch über die Pluginschnittstelle (siehe 3.7.2 auf Seite 81) nachgeladen werden.
Die Klassen, die über eine einzige Instanz überall im Quellcode von *Cupe* zu Verfügung stehen müssen (z.B. Programmparameter, metabolische Daten), sind unter der Kontrolle eines *Singletonmusters* programmiert worden.
Wie in Abbildung 3.3 auf Seite 82 in Zeile 6 zu sehen, kommt in *Cupe* auch die Kombination verschiedener Erzeugermuster vor.

Strukturmuster

Die Schnittstelle zu den Layoutalgorithmen der *AGD*-Bibliothek wurden gemäß dem *Adapter*-Muster implementiert. Dieses Strukturmuster ermöglicht es, die *AGD*-Bibliothek einfacher durch ihren Nachfolger, die *OGDF*-Bibliothek, zu ersetzen (siehe auch 2.1 auf Seite 23), weil sich die dazu nötigen Anpassungen im Wesentlichen nur auf eine Klasse (`Cupe_Layout_Interface`) erstrecken werden.
In *Cupe* werden Datencontainer und die Algorithmen, die auf ihnen operieren, über Iteratoren miteinander verbunden. Die bereits erwähnten *Funktionsobjekte* (2.4.5) kapseln dabei die speziell für *Cupe* entwickelten Operationen. Diese Art der Programmierung ermöglicht es, Schleifen als Template zu schreiben und lässt sich vom Compiler besonders gut optimieren. In *Cupe* kommen diese *Brücken*-Muster überaus häufig vor.

Verhaltensmuster

Alle Manipulationen am gezeichneten Graph lassen sich rückgängig machen und auch wiederherstellen. Um dieses zu ermöglichen wurde für jede Manipulation ein entsprechender *Befehl* implementiert. Diese Befehle werden von einer leistungsstarken Befehlsverwaltung kontrolliert (3.7.1).
Ein metabolisches Netzwerk ist innerhalb von *Cupe* in dreifacher Form repräsentiert: als interaktiver gezeichneter Graph, als *CUBIC-Matrix* (3.5.1) und als *LEDA-Graph*. Um all diese Datenstrukturen synchron zum gezeichneten Netzwerk zu halten, wird jedes *Cupe-Item* zum *Subject* mehrerer *Observer* definiert. Dieses *Beobachter*-Muster garantiert, dass alle Änderungen des Netzwerkes auf die entsprechenden Datenstrukturen übertragen werden.
Die neue Datenstruktur „*CUBIC Sparse-Matrix*“ (3.4.1) ist gemäß dem *Schablonen*-Muster implementiert worden. Hierdurch ist es möglich, beliebige Datentypen in der *CUBIC Sparse-Matrix* zu verwalten. Zusätzlich wurde der Zugriff über das *Iterator*-Modell implementiert, so dass die *CUBIC Sparse-Matrix* vollständig zur *Standard Template Library* kompatibel ist (2.2.2).

2.4.7 Abstrakte Datentypen

Als Datentyp versteht man die Zusammenfassung von Objekten (Daten) und der auf ihnen definierten Operationen (z.B. Addition) zu einer Einheit[1]. Sind die Eigenschaften eines Datentyps (z.B. schneller Zugriff auf seine Objekte) unabhängig von der Art seiner Objekte und die Definition der erlaubten Operationen unabhängig von ihrer konkreten Implementation, so spricht man von abstrakten Datentypen (im Gegensatz zu konkreten Datentypen). Für viele dieser Datentypen sind gemeinsame Operationen definiert (z.B. `sort()`, `push()`, `clear()`, usw. (s. Stroustrup, 2000). Dadurch ist es möglich, sie über eine gemeinsame Syntax anzusprechen, obwohl diese Operationen möglicherweise unterschiedlich implementiert sind. Während dieser Dissertation wurden Datentypen implementiert (z.B. der Datentyp *CUBIC-Sparse-Matrix*), die auf solchen abstrakten Datentypen aufbauen.

Balancierte Bäume als Assoziative Container

Ein assoziativer Container verwaltet Wertepaare. Unter Angabe eines Index, dem Schlüssel, kann auf den Inhalt, den abgebildeten Wert, zugegriffen werden. Man kann sich einen assoziativen Container auch als Array vorstellen, in dem der Index kein Integer sein muss. Auf den Schlüsseln dieser Wertepaare besteht eine Ordnung, so dass der kleinste Schlüssel in einer Baumstruktur (s. Abb. 2.1 auf der nächsten Seite) immer in der Wurzel steht. Üblicherweise werden assoziative Container als höhenbalancierte, binäre Suchbäume – beispielsweise AVL-Bäume (Ottmann & Widmayer, 1996) – organisiert. In einem solchen AVL-Baum ist die max. Höhe (vom Blatt zur Wurzel) auf $O(log(N+1))$ beschränkt (mit $O(x)$: asymptotische Komplexität

[1] In C++ spricht man von Klassen.

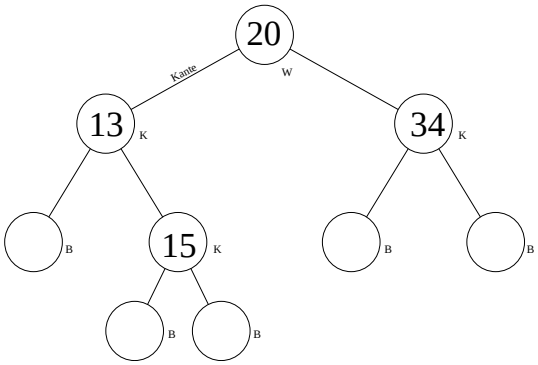


Abbildung 2.1: Binärer Suchbaum Bäume sind verallgemeinerte Listenstrukturen. Die Elemente eines Baumes heißen Knoten. Ein Knoten und sein Nachfolger sind über eine Kante miteinander verbunden. Der Knoten ohne Vorgänger heißt Wurzel (W), die Knoten ohne Nachfolger heißen Blätter (B). Alle Knoten, die nicht Blätter sind, heißen innere Knoten (K). Hat jeder innere Knoten genau zwei Nachfolger, spricht man von einem Binärbaum. Suchbäume sind geordnete binäre Bäume, die in den inneren Knoten Schlüssel speichern (hier die Werte: 13, 15, 20 und 34).

(Horowitz & Sahni, 1981) und N : Anzahl der Schlüssel, die im Baum gespeichert sind). Für die Praxis heißt das, dass in einer solchen Datenstruktur 65536 Schlüssel gespeichert werden können, auf die über maximal $log_2(65536) = 16$ Vergleiche (kleiner, gleich oder Blatt) zugegriffen werden kann. Assoziative Container werden in *Cupe* überaus häufig eingesetzt (z.B. um die Synonyme eines Enzymnamens schnell zu finden).

Hash-Tabellen

Besonders für zeitkritische Operationen, die einen „einfachen“ Schlüssel zu seinem assoziierten Wert abbilden, ist es häufig sinnvoll die Schlüssel nicht über Vergleiche zu referenzieren sondern direkt anzusprechen ($O(1)$ anstatt $O(log_2(N))$). Hierzu wird aus dem Schlüssel eine Prüfsumme – über die Hash-Funktion – generiert und der assoziierte Wert in einer Tabelle direkt unter dieser Prüfsumme abgelegt. Werden mehrere Schlüssel auf die gleiche Prüfsumme abgebildet, spricht man von einer Kollision. Die Kollisionsauflösung erfolgt über das Berechnen einer neuen Prüfsumme (rehashen). Bei steigender Belegungsdichte einer Hash-Tabelle kommt es vermehrt zu solchen Kollisionen und den mit ihnen einhergehenden Operationen zur Kollisionsauflösung. Für die Güte einer Hash-Tabelle ist demnach entscheidend Kollisionen zu vermeiden. Hash-Tabellen bilden die Grundlage der *CUBIC Sparse-Matrix*.

Stacks und Listen

Ein Stack (Kellerspeicher) speichert Elemente und gibt sie in umgekehrter Reihenfolge wieder aus. Hier werden Stacks vornehmlich dafür genutzt Befehle (2.4.6) in der richtigen Reihenfolge rückgängig zu machen oder wiederherzustellen.

Stacks werden im allgemeinen als sog. doppelt verkettete Listen implementiert. In solchen Listen hat jedes Element ein Paar von Zeigern. Von diesen zeigt einer zu dem Vorgänger und einer zu dem Nachfolger des Elements. Somit führen zu jedem Element zwei Zeiger hin und zwei Zeiger zu anderen Elementen (außer beim ersten und letzten Element). Da für Operationen auf solchen Listen meist nur die beteiligten Zeiger „umgebogen“ werden müssen, eignen sie sich besonders gut, um Elemente einzufügen, zu löschen oder zu verschieben. Die Liste der Elemente lässt sich darüber hinaus auch schnell bidirektional durchsuchen und sortieren. Listen werden in *Cupe* ebenfalls sehr häufig verwendet (z.B. um Metabolitnamen zu sortieren).

2.5 Bibliotheken

Ein so umfangreiches Projekt wie *Cupe* kann man nur unter der Zuhilfenahme von Programmbibliotheken realisieren. Diese Bibliotheken liefern die grafische Oberfläche, die Grundgerüste für Datenstrukturen und Algorithmen sowie nicht zuletzt auch die Layoutalgorithmen, die in *Cupe* Verwendung finden. Bibliotheken beschleunigen die Entwicklung und helfen Programmierfehler zu vermeiden. Die Benutzung von Bibliotheken schafft aber auch neue Abhängigkeiten (vergleiche 2.1 auf Seite 23) und erfordert zusätzliche Zeit, um sich mit ihrer effektiven Verwendung vertraut zu machen.

2.5.1 Standard Template Library

Die *Standard Template Library* (STL) ist eine C++-Bibliothek aus Containerklassen, Algorithmen und Iteratoren. Sie biete viele der grundlegenden Algorithmen und Datenstrukturen der Informatik. Zusätzlich ist die STL eine generische Bibliothek, das heißt, dass ihre Komponenten für verschiedene Datentypen parametrisiert werden können. Der überwiegende Teil der Entwicklungen (siehe 2.4.7 auf Seite 35) in *Cupe* ist unter der Verwendung der STL entstanden (Breymann, 2002). Diese Bibliothek ist fester Bestandteil jedes C++ Compilers und muss nicht zusätzlich installiert werden. Für diese Bibliothek gibt es seit 1998 einen ISO-Standard, so dass man davon ausgehen kann, dass der auf ihr beruhende Quellcode von allen aktuellen C++ Compilern übersetzt werden kann. Für die Entwicklung der CUBIC-Matrix (siehe 3.4.1 auf Seite 63) wurde zusätzlich der Datentyp *hash_map* benutzt. Dieser generische Datentyp gehört nicht zum Umfang der STL und stammt ursprünglich von der Firma der *Silicon Graphics Incorporated*. Mittlerweile wird die *hash_map* als Erweiterung zur STL mit der *GNU Compiler Collection* geliefert.

2.5.2 AGD

AGD (Alberts *et al.*, 1997) ist eine Bibliothek für das Zeichnen von Graphen, sie liefert eine große Menge etablierter und neuerer Algorithmen zum Zeichnen zweidimensionaler Graphen und liefert eine gute Schnittstelle, um neue Zeichenalgorithmen

zu implementieren. Diese Bibliothek liegt dem Quellcode von *Cupe* bei und wurde für die eigenen Bedürfnisse angepasst.

2.5.3 LEDA

LEDA (Näher & Mehlhorn, 1990b) ist eine Klassenbibliothek für effiziente Datenstrukturen und Algorithmen. Die Entwickler besitzen gründliche Fachkenntnisse im Bereich der Graphen- und Netzwerkprobleme und liefern in diesen Bereichen ausgezeichnete Algorithmen. *LEDA* wird von der *AGD*-Bibliothek(Alberts *et al.*, 1997) benötigt und liefert die Graph-Datenstrukturen in *Cupe*.

2.5.4 Qt

Qt (Trolltech, 2003) ist eine C++ Bibliothek zur Erstellung grafischer Anwendungen. Diese Bibliothek wird von der Firma Trolltech entwickelt und ist unter einer freien Lizenz für nicht kommerzielle Programme verfügbar. Sie ist vollständig objektorientiert und kann sowohl für Windows als auch Linux eingesetzt werden. *Cupe* benutzt diese Bibliothek für die grafische Benutzerschnittstelle.

2.5.5 Boost

Boost (u.a. Karlsson, 2005) ist eine Sammlung von freien C++-Bibliotheken, die ein breites Spektrum an portablen Problemlösungen bietet. Die Bibliothek wird von sehr vielen Entwicklern unterstützt und in vielen C++-basierten Projekten benutzt. Es ist beabsichtigt, dass mit der weiteren Etablierung unter den Entwicklern eine dann ausgereifte Variante zum C++-Standard erhoben werden soll, oder zumindest ausgereifte Teile von Boost in die C++-Standardbibliothek aufgenommen werden. In *Cupe* wird die *Boost*-Bibliothek beispielsweise für das Speichermanagement, Parser oder neue Datenstrukturen benutzt.

2.5.6 OpenGL

OpenGL (Open Graphics Library, OpenGL.org, 2005) ist eine Spezifikation für ein plattform- und programmiersprachenunabhängiges API (Application Programming Interface) zur Entwicklung von 3D-Computergrafik. Die Implementierung des OpenGL-API gibt es als Open Source-Implementierungen in der Mesa-Bibliothek oder auch als speziell vom Grafikkarten-Hersteller angepasste Version. Diese Bibliothek kann optional zu *Cupe* gebunden werden, um die dreidimensionale Molekülstruktur eines Metaboliten darzustellen.

3 Ergebnisse

Der neue *Cubic Pathway Editor* (*Cupe*) verfügt über eine vollständige grafische Bedienoberfläche und bietet zahlreiche Methoden für die automatische oder manuelle Erstellung und Analyse von Stoffwechselnetzwerken. Über die AGD-Bibliothek (siehe 2.5.2 auf Seite 37) können für die mit *Cupe* erstellten Graphen die jeweils passenden Zeichungen automatisch berechnet werden. Die Verwaltung der großen Datenmengen erfolgt über die eigens neu entwickelte Datenstruktur *Cubic-Matrix* und den dazugehörigen Operationen. Die Weiterentwicklung von *Cupe* ist über eine leistungsstarke Plugin-Schnittstelle möglich und kann auf der eigenen Internetseite besonders leicht dokumentiert werden.

3.1 Benutzerschnittstelle

Die Benutzerschnittstelle von *Cupe* ist zweigeteilt (vergleiche Abbildung 3.1 auf der nächsten Seite). Das Editorfenster bietet die gesamte Funktionalität, die man auch von modernen Anwendungen kennt. Es existieren Menüs und Werkzeugleisten, um Graphen zu zeichnen und zu speichern. Graphen können aus unterschiedlichen Dateiformaten importiert und als Bilder exportiert werden (siehe 3.1.4 auf Seite 49). Die Menüs und Werkzeugleisten sind dynamisch und können z.B. von Plugins erweitert werden, um weitere Analysemethoden bereitzustellen. Die Datenbankschnittstelle bietet alle Methoden, um automatisch Informationen zu den ausgewählten Molekülen anzuzeigen und Netzwerke aus beliebigen Reaktionen zusammenzustellen.

3.1.1 Der Editor

Die weiße Fläche im Hauptfenster ist die Zeichenfläche; dieses *Board* wird von einem *Editor-Modul* kontrolliert (siehe Abbildung 3.2 auf Seite 41). Der *Editor* hat die Aufgabe, alle Mausbewegungen aufzubereiten und an die entsprechenden Elemente (*Cupe-Item*) auf dem Board weiterzuleiten. Der Editor wertet diese *Mausevents* in der Regel nicht aus. Welche Reaktion z.B. auf das Drücken der rechten Maustaste erfolgen muss, wird von den Elementen, die es betrifft, selbst bestimmt. Erfolgt der Rechtsklick etwa in einem freien Bereich der Zeichenfläche, wird das *Kontextmenü* des Boards aufgerufen, und man könnte unter anderem den Namen des Boards ändern. Wird hingegen die rechte Maustaste bedient, wenn sich der Mauszeiger über einem Knoten befindet, wird stattdessen das Kontextmenü für diesen Knoten aufgerufen, und es wird beispielsweise angeboten, ihn zu löschen. Die algorithmische Trennung zwischen der Detektion und der Auswertung eines Mausevents,

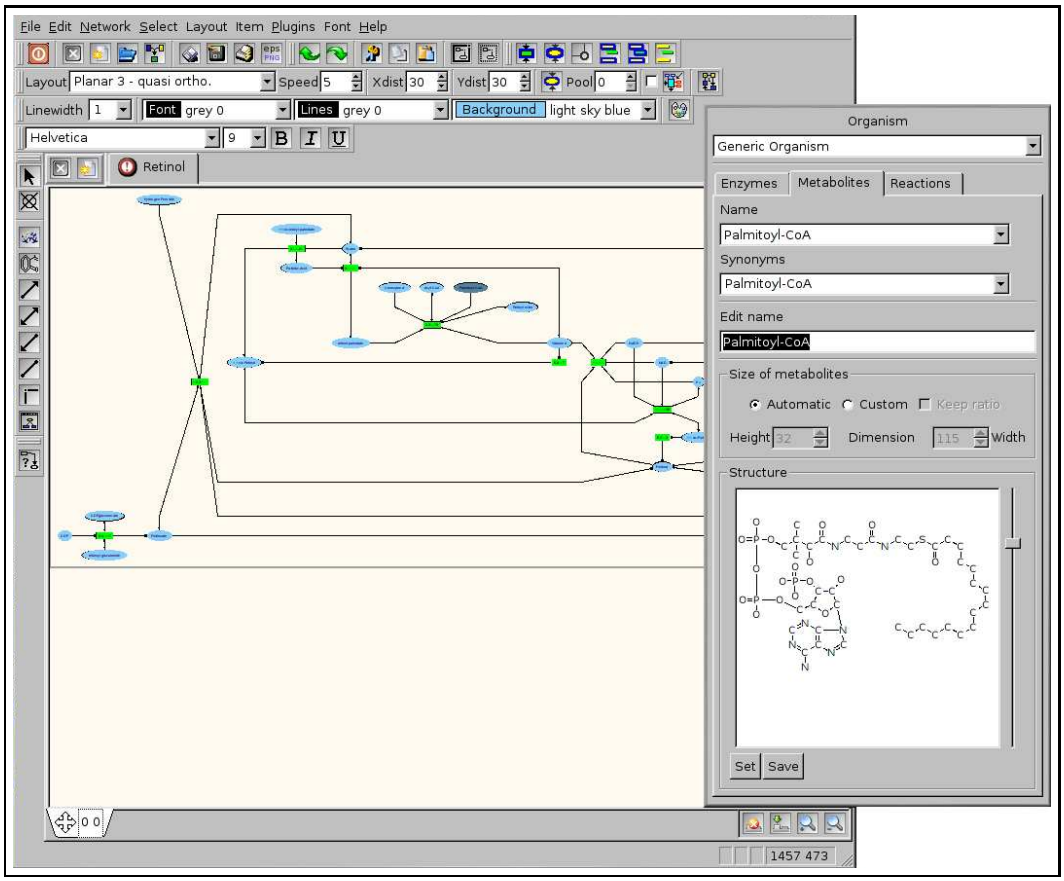


Abbildung 3.1: Überblick Die Oberfläche von *Cupe* besteht aus zwei Elementen: dem Hauptfenster (links) und der Datenbankschnittstelle (rechts)

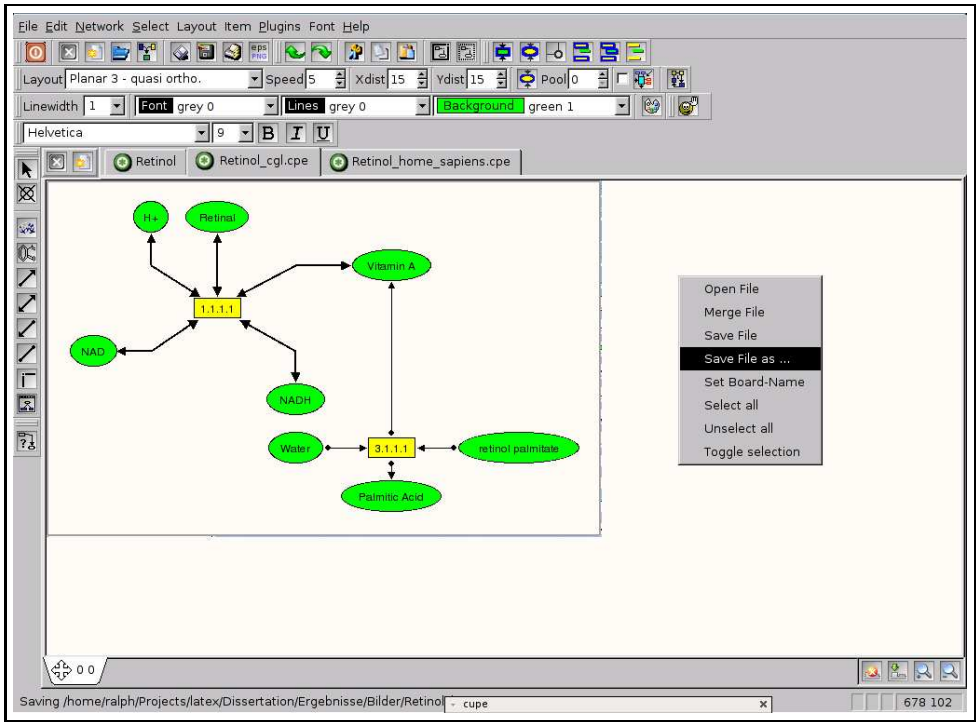


Abbildung 3.2: Das Editormodul Die Zeichenfläche des Hauptfensters wird Board genannt. Jedes Board wird über ein Editormodul kontrolliert. Der Editor wertet die Mausereignisse aus und leitet sie an die entsprechenden Objekte auf dem Board weiter. Der graue Rahmen zeigt die Grenzen der aktiven Zeichenfläche an. Diese Grenzen können mit dem Mauszeiger interaktiv verschoben werden. Zusätzlich bietet jeder Editor ein Kontextmenü, um z.B. den Boardnamen zu setzen.

ermöglicht es, neue Graphenelemente zu entwickeln, ohne dass die Implementierung des Editor-Moduls angepasst werden muss. Von dieser allgemeinen Behandlung der Mausevents weicht nur das *Selection-Management* ab. Das heißt, dass der Editor das Zusammenstellen von Graphenelementen zu Gruppen selber steuern kann. Ob aber ein Cupe-Item überhaupt ausgewählt oder einer Gruppe zugeordnet werden kann, wird jedoch vom Cupe-Item selber entschieden. Wenn das nicht möglich ist, kann der Editor während dieser Operation nicht auf dieses Element zugreifen.

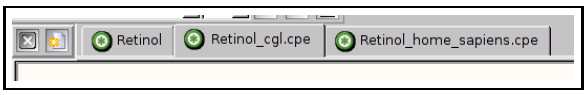


Abbildung 3.3: Mehrere Editoren Innerhalb eines Hauptfensters können mehrere Editoren (Boards) gleichzeitig geöffnet werden.

Mehrere Editoren öffnen

Innerhalb eines Hauptfensters können beliebig viele Boards gleichzeitig verwaltet werden (vergleiche Detailausschnitt 3.3 auf der vorherigen Seite). Auf diese Weise können mehrere Graphen zum Vergleich geöffnet werden. Plugins können die Ergebnisse ihrer Analysen auf neuen Zeichenflächen darstellen. Jeder Editor verfügt über ein eigenes *Command-Management*, das heißt die Funktionen „Rückgängig“ und „Wiederherstellen“ von *Befehlen* (z.B. Bewegen eines neuen Knotens) sind für jeden Editor spezifisch. Zudem wird auch stets sichergestellt, dass ein und derselbe Graph nicht in zwei unterschiedlichen Editoren geöffnet werden kann.

Mehrere Ansichten öffnen

Wenn mehrere Ansichten eines Biosynthesenetzwerks nötig sind, dann können diese innerhalb eines Boards geöffnet werden (siehe Abbildung 3.4 auf der nächsten Seite). Die Manipulationen in einer Ansicht werden dabei von allen anderen Ansichten übernommen, so dass der Graph jederzeit synchron zu den aktuellen Änderungen gehalten wird. Durch diese Technologie ist es möglich, in sehr großen Graphen bequem zu navigieren und jeweils die Ausschnitte zu fokussieren, die Gegenstand genauerer Untersuchungen sein sollen. Jede Ansicht kann zudem noch in der jeweils angepassten Vergrößerungsstufe betrachtet werden. Etwaige Mausevents werden entsprechend des jeweiligen *Zoomfactors* umgerechnet und zu dem richtigen Cupe-Item weitergegeben.

Editorlupe

Eine besondere Übersicht über große Graphen ermöglicht die *Lupe* (siehe Abbildung 3.5 auf der nächsten Seite). Die Lupe wird aufgerufen, indem man zuerst die *Leertaste* und dann die *linke* Maustaste drückt. In diesem Moment wird der Ausschnitt, über dem sich der Mauszeiger befindet, in einer zweifachen Ausschnittsvergrößerung dargestellt. Man kann die Leertaste nun wieder freigeben und die Lupe mit gedrückter Maustaste über das Netzwerk bewegen. Diese Technik funktioniert unabhängig von der Vergrößerungsstufe der gewählten Ansicht. Wenn ein großer Graph eines Reaktionsnetzwerks bereits stark verkleinert dargestellt wird, ist der automatisch gewählte Faktor der Lupenvergrößerung unter Umständen nicht ausreichend, um die Beschriftung der Knoten zu erkennen. In diesem Fall kann, solange die Lupe aktiviert ist, über die Tasten „+“ und „−“ die Vergrößerungsstufe der Lupe angepasst werden.

3.1.2 Netzwerke manuell konstruieren

Mit Hilfe der linken senkrechten Werkzeugleiste (Abbildung 3.6 auf Seite 44) kann man eigene Netzwerke zeichnen. Insgesamt werden zehn unterschiedliche Aktionen angeboten. Jede dieser Aktionen versetzt den Editor, der das Board kontrolliert, in einen entsprechenden Zustand. Es ist möglich, den Editor in den sogenannten *Point-*

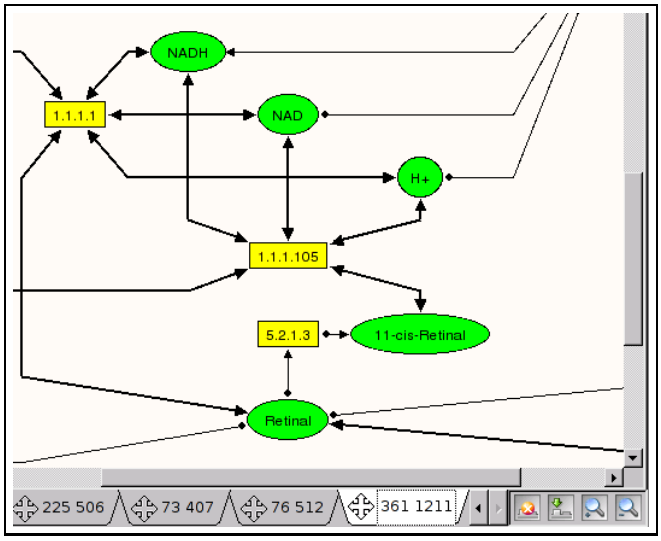


Abbildung 3.4: Mehrere Ansichten eines Boards Innerhalb eines Boards können mehrere Ansichten auf denselben Graphen geöffnet werden. Diese Ansichten können auch in unterschiedlichen Vergrößerungsstufen dargestellt werden (siehe Icons rechts).

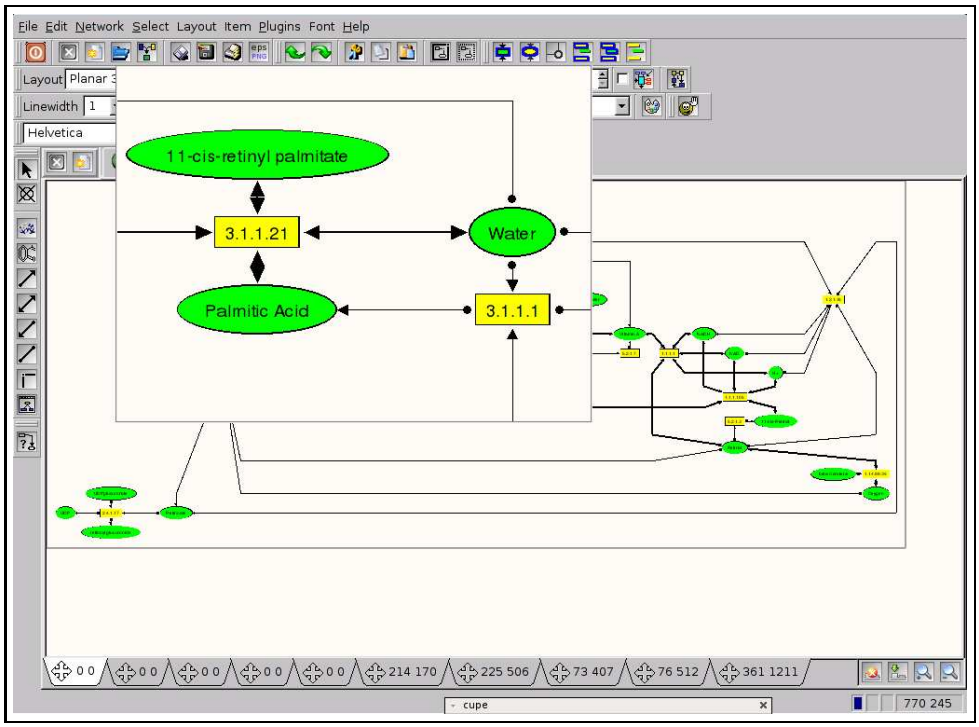


Abbildung 3.5: Die Lupe Große Graphen, die z.B. verkleinert dargestellt werden, können mit einer Lupe betrachtet werden.

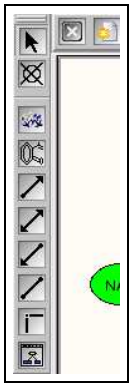


Abbildung 3.6: Netzwerkelemente Von oben nach unten: Anzeigen bzw. markieren von Elementen, Löschen von Elementen, Enzyme einfügen, Metabolite einfügen, rechts-, links-, bidirektional gerichtete und ungerichtete Kanten einfügen, Stützpunkte für Kanten einfügen und Molekülcluster einfügen. Um ein Netzwerk manuell zu konstruieren, werden die Fabriken über ihre Schaltflächen ausgewählt und dann erst ihre Eigenschaften definiert.

oder *Delete*-Modus zu versetzen, um Elemente zu markieren oder zu löschen. Für das Einfügen von Netzwerkelementen wird der zugeordnete *Insert*-Modus versetzt. Um z.B. die einfache Reaktion A $\overset{E}{\rightleftharpoons}$ B zu zeichnen, aktiviert man die Schaltfläche zum Einfügen von Metaboliten und klickt anschließend zweimal auf einen freien Bereich auf der Zeichenfläche. Nun wechselt man das Werkzeug und wählt die Enzymschaltfläche aus. Durch einen einfachen Linksklick auf dem Board wird ein neues Enzym eingeführt. Um die Kanten zwischen den Molekülen zu zeichnen, wählt man die entsprechende Schaltfläche aus und klickt mit dem Mauszeiger zuerst auf das eine und dann auf das andere der beiden Moleküle, die mit einer Kante verbunden werden sollen. Wenn man möchte, dass die Elemente des Graphen abweichend von der Standardeinstellung gezeichnet werden, kann man die Farben, Linien und die Schriftart im Hauptfenster neu einstellen und anschließend gemäß dieser Vorgaben zeichnen. Diese neuen Einstellungen werden beibehalten, bis sie wieder geändert werden und gelten auch zwischen verschiedenen Programmstarts als neue Standardeinstellung. Es ist auch möglich, zuerst das gesamte Reaktionsnetzwerk zu zeichnen und dann die Graphenelemente auszuwählen bzw. zu gruppieren und gleichzeitig neu zu formatieren. Alle diese Aktionen (*Befehle*) können dabei jederzeit wieder rückgängig gemacht werden. *Cupe* macht auch keine Vorgaben, welche Art von Netzwerk gezeichnet werden soll. So ist es möglich, Metabolitnetzwerke, Enzymnetzwerke und Reaktionsnetzwerke, oder eine beliebige Kombination aus ihnen, zu zeichnen (vergleiche 1.3.2 auf Seite 21).

Enzyme

Immer dann, wenn die Enzymschaltfläche zum Einfügen von Enzymen aktiviert oder wenn ein bereits gezeichnetes Enzym markiert wird, werden alle bekannten Informationen zu diesem Molekül automatisch angezeigt (siehe Abbildung 3.7 auf der nächsten Seite). Über diese Anzeige werden die bekannten Synonyme zu einem Enzymnamen und die EC-Nummer angezeigt. Wenn es für die jeweilige Aussage wichtig ist, kann man problemlos auch eigene Namen vergeben. Zudem kann man

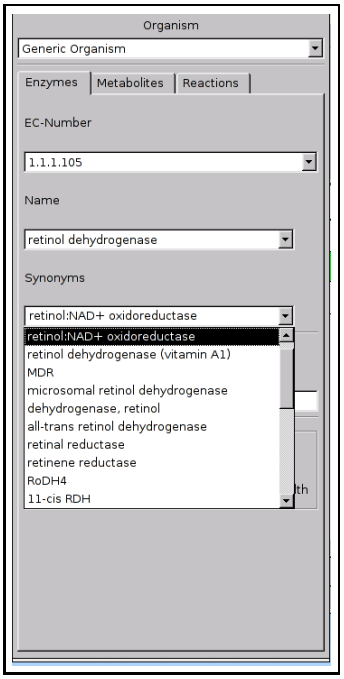


Abbildung 3.7: Enzyme Werden im Editor Enzyme eingefügt oder selektiert, dann werden die enzymspezifischen Informationen in diesem Dialog angezeigt. Über diesen Dialog können auch die alternativen Enzymnamen ausgelesen oder neue Namen vergeben werden. Darüber hinaus kann hier zwischen automatischer und manuell festgelegter Knotengröße gewechselt werden.

einstellen, ob ein Enzymknoten automatisch die richtige Knotengröße annimmt oder ob der Knoten einer anderen Strategie folgt. Das ist besonders wichtig, wenn Plugins die Knotengröße zur Darstellung erweiterter Informationen nutzen möchte (z.B. Konzentration während der Wachstumsphase der Zelle).

Metabolite

Die Informationen, die zu ausgewählten bzw. neu einzufügenden Metaboliten, präsentiert werden, sind analog zu denen der Enzyme (siehe Abbildung 3.8 auf der nächsten Seite). Zusätzlich zu den Namen und der Wahl zwischen automatischer und manueller Einstellung der Knotengröße, wird zusätzlich die Lewisstruktur des Metaboliten angezeigt. Wenn man die *OpenGL*-Bibliothek mit einbindet (siehe 2.5.6 auf Seite 38), ist sogar eine einfache dreidimensionale Molekülдarstellung möglich.

3.1.3 Netzwerke automatisch konstruieren

Neben der völlig freien Netzwerkkonstruktion ist es möglich, metabolische Netzwerke durch freie Auswahl aus der Reaktionsdatenbank zu erstellen. Der Anwender wird dabei durch eine grafische Oberfläche gut unterstützt (siehe Abbildung 3.9). Über diese Oberfläche ist es z.B. möglich, Reaktionen bestimmter Enzyme für ausgewählte Substanzen und für einen spezifischen Organismus von insgesamt ca. 4000 auszuwählen. Diese Auswahl kann dann über Mengen-Operationen beliebig erwei-

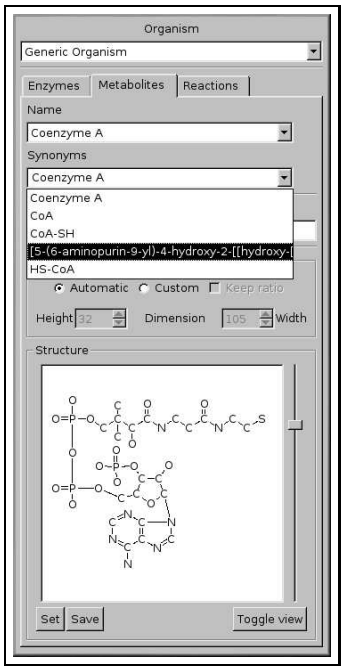
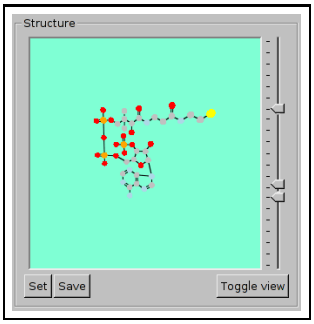


Abbildung 3.8: Metabolite Wenn im Editor Metabolite eingefügt oder selektiert werden, dann wird die Lewisstruktur des Metaboliten als 2D- oder 3D-Grafik (OpenGL) angezeigt. Über diesen Dialog können die alternativen Metabolitnamen ausgelesen oder neue Namen vergeben werden. Darüber hinaus kann hier auch zwischen automatischen oder manuellen Bestimmung der Knotengröße gewechselt werden.



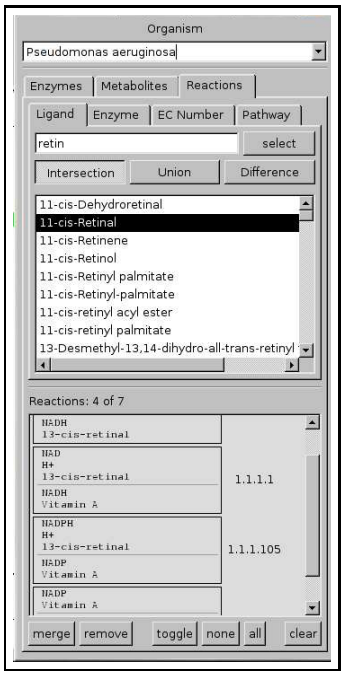


Abbildung 3.9: Netzwerke automatisch konstruieren Über dieses Menü ist es möglich, ein Netzwerk durch die Kombination beliebiger Reaktionen völlig frei zu konstruieren. Die Abbildung zeigt, dass von sieben bekannten Reaktionen mit Retinal vier für den Organismus *Pseudomonas aeruginosa* annotiert sind. Diese Reaktionen werden im unteren Drittel des Menüs angezeigt und können in den Graphen eingefügt werden. Über die Schaltflächen „Intersection“, *Difference* und *Union* können weitere Reaktionen mit dieser Auswahl kombiniert werden.

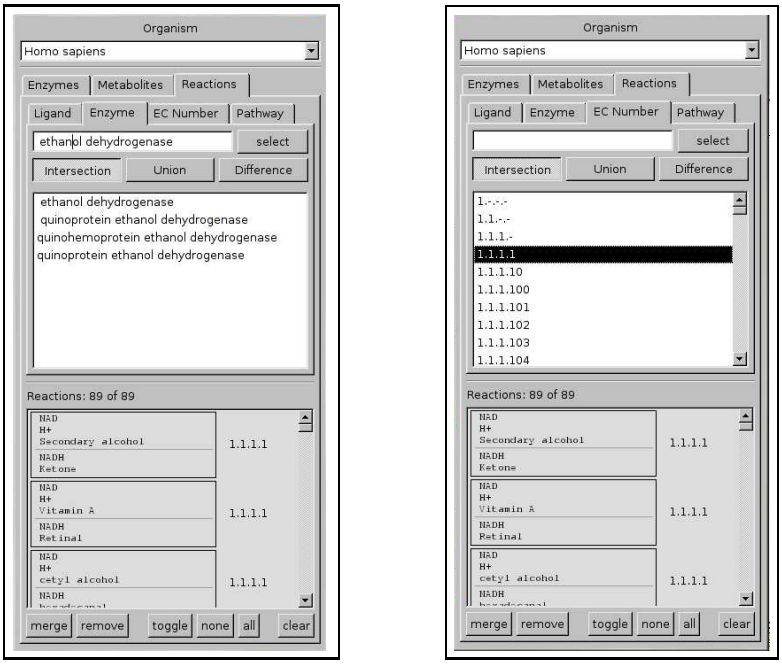


Abbildung 3.10: Auswahl von Reaktionen anhand von Enzymen Die Auswahl von Reaktionen kann anhand ihrer Enzyme erfolgen. Zur Auswahl stehen sowohl die Enzymnamen als auch die EC-Nummern.

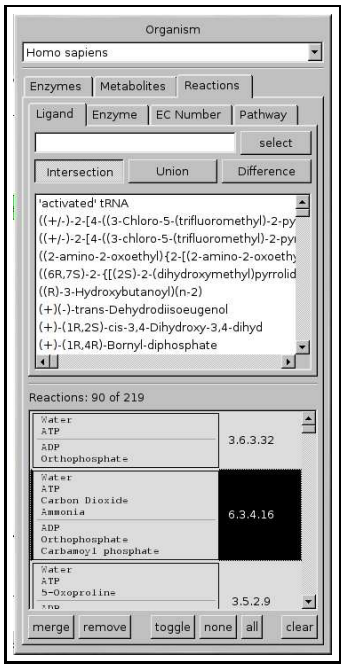


Abbildung 3.11: Metabolite Die Auswahl von Reaktionen anhand der Metabolite. Ausgewählt sind alle Reaktionen, an denen ATP, Wasser und ADP teilnehmen.

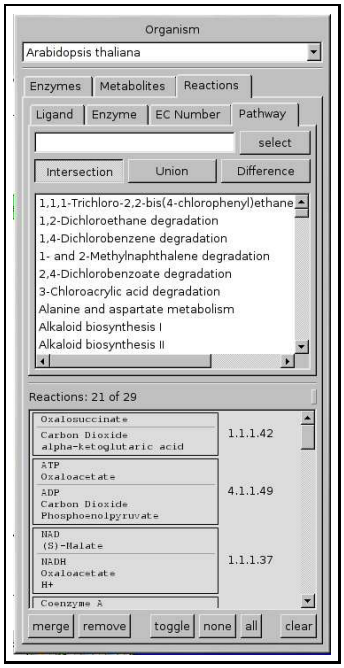


Abbildung 3.12: Reaktionen einer Stoffwechselkarte Reaktionen können in Pathways zusammengefasst werden. Diese Pathways können ebenfalls beliebig kombiniert werden. Die Abbildung zeigt die Auswahl der Reaktionen des Citratstoffwechsels.

tert oder eingeschränkt werden. Es existieren auch bereits vorgefertigte Sammlungen von Reaktionen für über 300 Biosyntheseprozesse.

Enzyme

Um Reaktionen gemäß der sie katalysierenden Enzyme auszuwählen, gibt es zwei Möglichkeiten. Es ist sowohl möglich, die Enzyme anhand ihrer Namen zu suchen als auch entsprechend ihrer EC-Nummern (siehe Abbildung 3.10 auf der vorherigen Seite). Dabei wird der Anwender von einer Auswahlliste unterstützt, die immer nur die Enzyme anzeigt, die zu dem bereits eingegebenen Suchwort passen. Der Suchraum wird also inkrementell verkleinert und nur noch ähnliche Ergebnisse werden angeboten. Dadurch ist es nicht notwendig, die genaue oder vollständige Bezeichnung eines Enzyms einzugeben, um es zu finden.

Metabolite

Über die grafische Datenbankschnittstelle ist es auch möglich, Reaktionen anhand der beteiligten Metabolite zu suchen. Die Benutzung dieses Dialogs wird, wie bei der Suche anhand der Enzyme, über eine inkrementelle Suche erleichtert. Durch die Kombination unterschiedlicher Suchen können auch komplexere Anfragen beantwortet werden (siehe Abbildung 3.11 auf der vorherigen Seite).

Pathways

Als besonders nützlich hat sich die Auswahl nach Pathways erwiesen. So lassen sich z.B. leicht alle Reaktionen des *Citratstoffwechsels* anzeigen (siehe Abbildung 3.12 auf der vorherigen Seite). Durch die Verknüpfung mit weiteren Abfragen lassen sich auch mehrere Stoffwechselpfade kombinieren.

Liste der Reaktionen anzeigen

Im Zuge der Entwicklung von *Cupe* hat es sich herausgestellt, dass die zusammengestellten Reaktionen auch als Textdatei nützlich sein können; zum einem, um die Datenbestände zu verifizieren, zum anderen aber, um die Details der Reaktionen auch in anderen Dokumenten wiederzugeben. Zu diesem Zweck kann die Liste der Reaktionen ausgegeben und in einem voll funktionsfähigen Editor bearbeitet werden (siehe Abbildung 3.13 auf der nächsten Seite). Diese Funktionalität ist bisher aber noch komplett implementiert, so dass sie nur über eine provisorische Schaltfläche zu erreichen ist. In der Abbildung 3.12 kann man die kleine unbeschriftete Schaltfläche am rechten Rand neben dem Schriftzug „*Reactions: 21 of 29*“ erkennen.

3.1.4 Sonstige Dialoge

Die Funktionalität einer grafischen Anwendung ergibt sich fast ausschließlich aus den bereitgestellten Dialogen. In *Cupe* gibt es noch eine Vielzahl von weiteren Dialogen, die bisher noch nicht besprochen worden sind. Die wichtigsten werden in den folgenden Absätzen kurz vorgestellt.

Export und Import

Cupe bietet einen komfortablen Dateidialog. Dateien, die bereits geöffnet waren, können am schnellsten aus der Liste „*Recent files*“ ausgewählt und geladen werden (siehe Abbildung 3.14, links). Neue Netzwerke können entweder aus Dateien im GML-Format (*.gml) importiert oder aus dem *Cupe*-eigenen XML-Format („*.cpe“) eingelesen werden. Wird der Graph eines metabolischen Netzwerkes bearbeitet, kann er anschließend im *Cupe*-Dateiformat (*.cpe) gespeichert werden. Um Abbildungen des Netzwerkes auch in anderen Dokumenten darzustellen, können die Graphen auch als Bitmap-Datei im Format *Portable Network Graphic* (*.png) oder als echte Vektorgrafik im Format *Encapsulated PostScript* (*.eps) exportiert werden.

Auswahl

Bei der Arbeit mit Netzwerken kommt es schnell vor, dass man gleichartige Elemente eines Graphen auch auf die gleiche Art behandeln möchte. Zu diesem Zweck bietet *Cupe* viele vordefinierte Makros an, um die Selektion von Enzymen, Metaboliten, Clustern und gepoolten Molekülen zu vereinfachen (siehe 3.14, rechts oben).

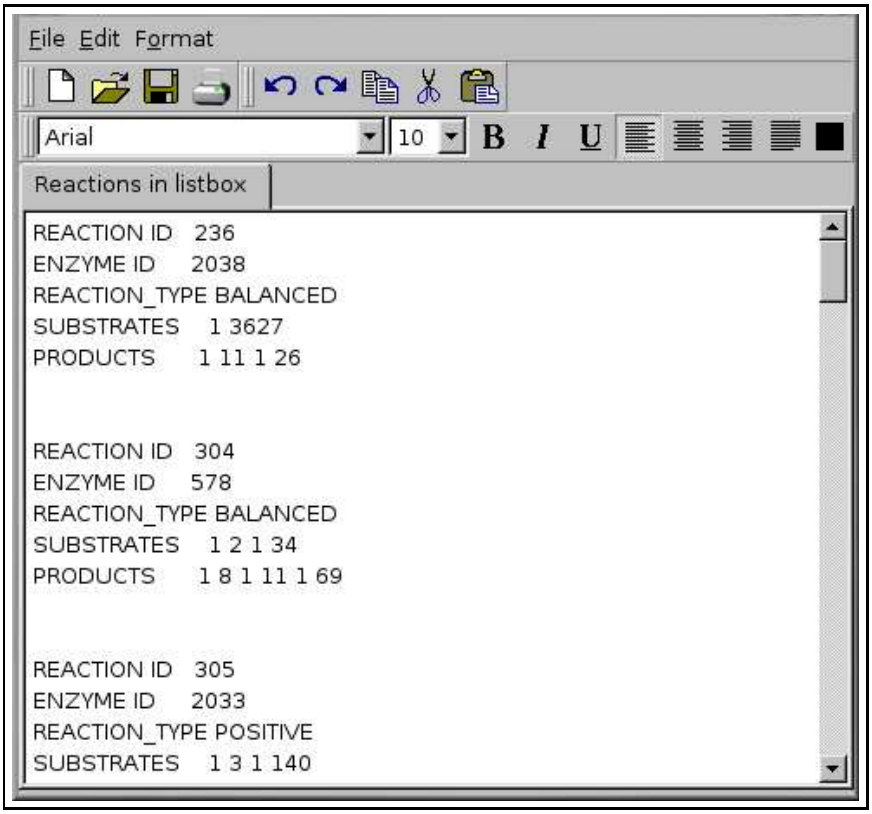


Abbildung 3.13: Liste der Reaktionen Die Liste der Reaktionen kann über eine versteckte Schaltfläche ausgegeben werden. Diese Schaltfläche ist in der Abbildung 3.12 auf Seite 48 rechts neben der Beschriftung *Reactions: 21 of 29* zu erkennen.

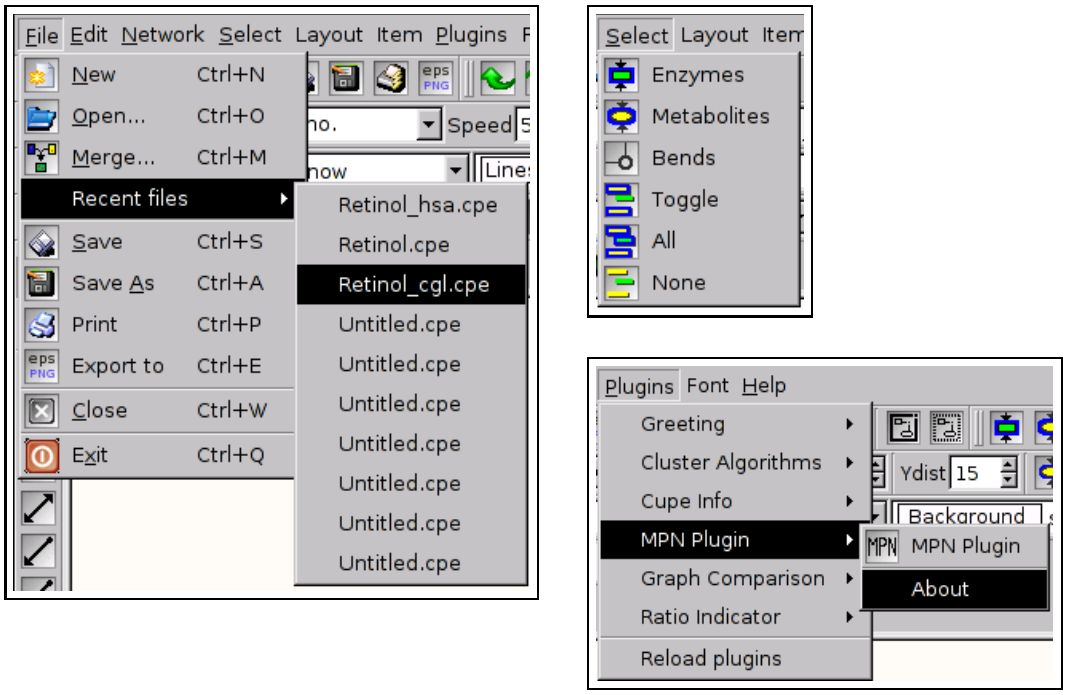


Abbildung 3.14: Weitere Dialoge Cupe verfügt über eine vollständige Schnittstelle für den Datei-Import und Datei-Export (links). Unterstützt wird sowohl ein eigenes Dateiformat als auch der Export in unterschiedliche Bildformate. Die Auswahl der Netzwerkelemente wird über eine Vielzahl an vordefinierten Makros erleichtert (oben rechts). Cupe kann über eine Pluginschnittstelle um weitere Analysemethoden und Darstellungsmöglichkeiten erweitert werden (unten rechts).

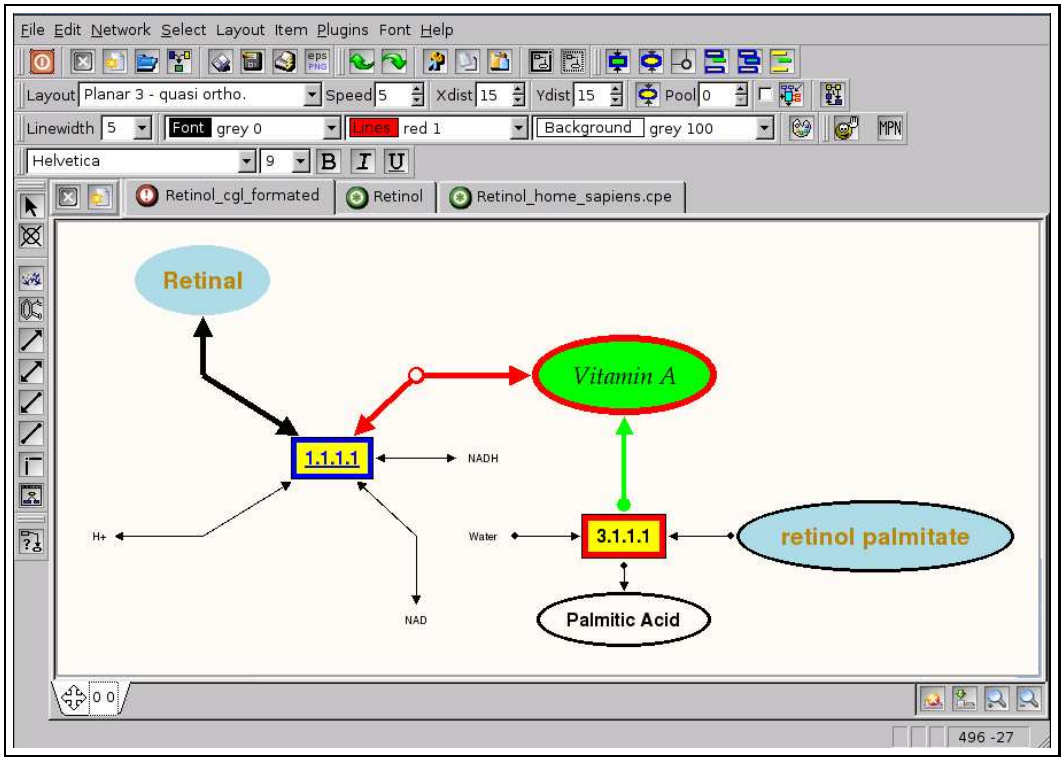


Abbildung 3.15: Manuelle Formatierung des Netzwerks Ein Netzwerk kann je nach Bedarf formatiert werden. Es stehen alle Möglichkeiten zur Verfügung, die man auch von anderen modernen grafischen Anwendungen kennt. Zur Verdeutlichung wird ein Stützpunkt auf einer Kante dargestellt (1.1.1.1 $\longrightarrow$ *Vitamin A*). Diese Stützpunkte sind nur während der direkten interaktiven Manipulation mit dem Mauszeiger sichtbar.

Plugins

Plugins werden in ein eigenes Menü eingetragen (siehe 3.14, rechts unten). Die Einträge in diesem Menü sind dynamisch und hängen natürlich von der Verfügbarkeit unterschiedlicher Plugins ab. Zu jedem Plugin gibt es einen einfachen Informationsdialog, der ebenfalls im *Cupe*-Pluginmenü abgelegt wird.

3.2 Manuelle Formatierung

Bei der Erzeugung und Analyse von metabolischen Netzwerken möchte man Unterschiede zwischen den Elementen des Netzwerkes grafisch darstellen. Angefangen bei der Farbe des Hintergrunds von Knoten bis zur geeigneten Wahl der Schriftart und ihrer Attribute für die Beschriftung, gibt es in *Cupe* viele Möglichkeiten, um solche Hervorhebungen anzubringen.

3.2.1 Knoten

Zu den Knoten in einem gezeichneten Netzwerk von *Cupe* zählen Metabolite, Enzyme, die Stützpunkte der Kanten und Cluster. Diese Knoten können über eine gemeinsame Schnittstelle formatiert werden. Hierzu werden sie zunächst einzeln oder als Gruppe selektiert, um ihnen dann über die dazugehörigen Menüs neue Attribute zuzuweisen.

Enzyme und Metabolite

Enzyme und Metabolite werden auf die gleiche Weise editiert. Es ist möglich, die Umrandung auszublenden oder besonders dick zu zeichnen. Man kann die Vorder- und Hintergrundfarbe festlegen und die Schriftart und ihre Eigenschaften (fett, kursiv, unterstrichen) bestimmen (siehe Abbildung 3.15 auf der vorherigen Seite).

Stützpunkte

Stützpunkte sind die Punkte, die die Knicke in Kanten festlegen. Man kann sie neu einfügen, auswählen und verschieben. Ihre Dicke hängt aber von der Kantendicke und ihre Farbe von der Kantenfarbe ab (vergleiche Abbildung 3.15). Wenn sie nicht ausgewählt sind, werden sie im Graphen des Reaktionsnetzwerks nicht sichtbar dargestellt.

Cluster

Cluster sind Superknoten, die ihrerseits Knoten – auch Cluster – aufnehmen können (siehe 3.16 auf der nächsten Seite). Ein Cluster stellt eine eigenständige Zeichenfläche dar und kann den Graphen, den er enthält, unabhängig vom Gesamtgraphen ausrichten. Ebenso ist das Layout innerhalb eines Clusters unabhängig vom Layout des Graphen, der ihn umgibt. Zu den Formatierungsmöglichkeiten der allgemeinen Knoten können Cluster gänzlich unsichtbar gemacht werden. Sie sind dann nur sichtbar, wenn sie als Knoten ausgewählt werden. Da Cluster als eigenständige Zeichenfläche innerhalb eines Graphen gelten, ist die Größe der Fläche, die sie verwalten, ebenfalls dynamisch. Sie richtet sich nach der Größe des Graphen, den der Cluster umschließt und kann – sofern die Grenzen nicht unsichtbar sind – mit den Mauszeiger verändert werden (vergleiche Abbildung 3.2 auf Seite 41).

3.2.2 Kanten

Kanten können gerichtet oder ungerichtet gezeichnet werden. Es können neue Stützpunkte in sie eingefügt und auch wieder gelöscht werden. Kanten können an ihren Endpunkten „angefasst“ und mit anderen Knoten verbunden werden. Es ist nicht möglich, Kanten mit gleichem End- und Startmolekül zu erzeugen. Auch können Kanten zur Zeit nicht mit der Maus ausgewählt werden. Wenn man eine Kante in

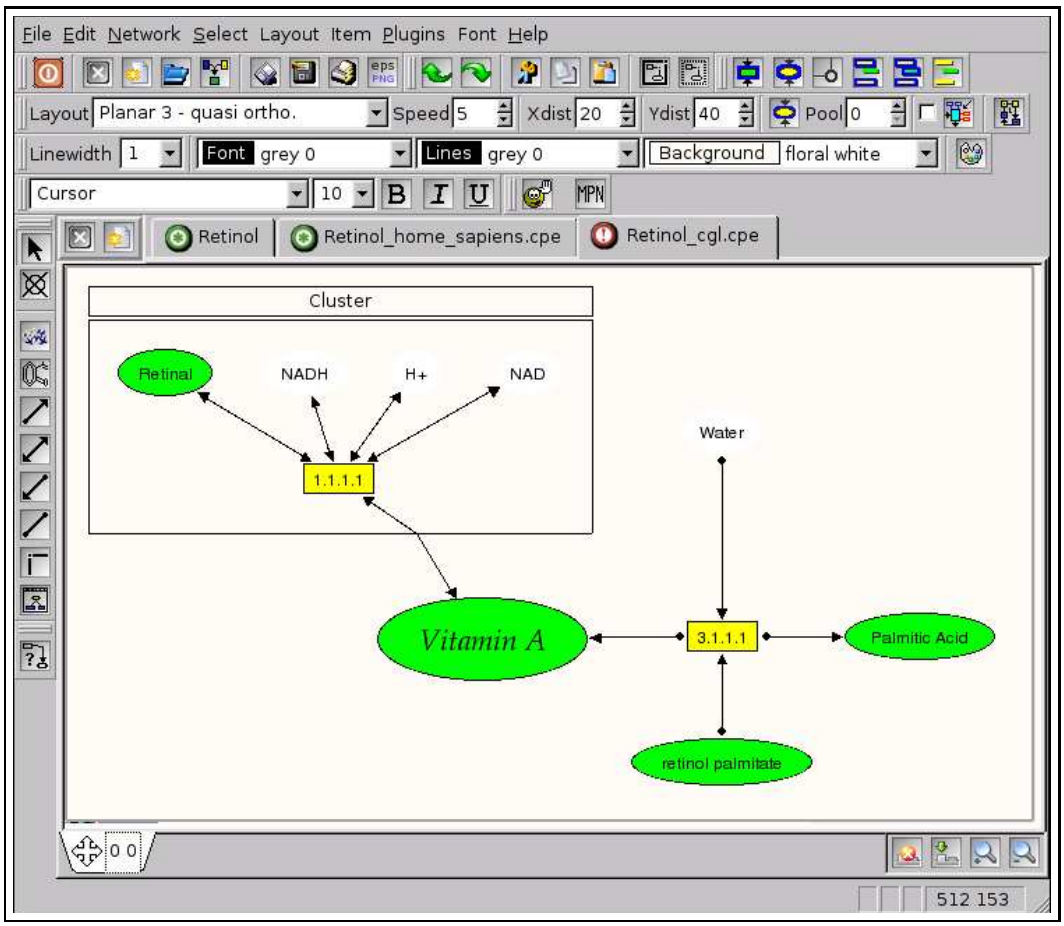


Abbildung 3.16: Netzwerkcluster Netzwerkelemente können in Gruppen, den sogenannten Clustern, zusammengefasst werden. Diese Cluster können nach Belieben formatiert und unabhängig vom Gesamtgraphen ausgerichtet werden (Graphlayout). Cluster können auch kollabiert und unsichtbar sein sowie wiederum andere Cluster enthalten (siehe Abbildung 3.25 auf Seite 60).

einer anderen Farbe oder Strichstärke benötigt, so muss man sie neu zeichnen. Algorithmische Zugriffe über Plugins und andere direkte Zugriffe können eine Kante ändern, ohne sie neu zu zeichnen.

3.2.3 Auswahl

Die Knoten in einem Graphen können einzeln mit dem Mauszeiger und Drücken der linken Maustaste ausgewählt werden. Wenn man mehr als einen Knoten auswählen möchte, dann kann man durch gleichzeitiges Niederhalten der „*Strg*“-Taste weitere Knoten auswählen. Solange der Editor im Modus *Anzeigen* ist, ist es möglich, durch gleichzeitiges Drücken der linken Maustaste und Bewegen des Mauszeigers eine rechteckige Auswahl zu treffen. Die Möglichkeit, einen rechteckigen Bereich im metabolischen Netzwerk auszuwählen, existiert auch für den Modus *Löschen*. Die so ausgewählten Elemente werden dann gelöscht, sobald die linke Maustaste gelöst wird.

3.3 Automatisches Zeichnen

Die Zeichnung eines Stoffwechselgraphens wird mit Hilfe der umfangreichen AGD-Bibliothek berechnet (2.5.2). Für *Cupe* wurde diese Bibliothek geringfügig angepasst und dem Sourcecode beigelegt. Die ausschlaggebenden Vorteile, die zur Verwendung von AGD führten, liegen darin, dass AGD in der Programmiersprache C++ geschrieben ist und dass zu den Entwicklern von AGD während der gesamten Arbeit ein sehr guter Kontakt bestand. AGD benutzt die kommerzielle LEDA-Bibliothek und verfügt so über professionell entwickelte Algorithmen und Datenstrukturen zur Graphendarstellung. Mit der immer wieder neuen Bereitstellung aktueller C++-Compiler und C++-Bibliotheken hat sich aber die Abhängigkeit zu kommerziellen Softwarepaketen als hinderlich für die weitere Forschung im akademischen Umfeld erwiesen. Daher wurde mit der Entwicklung einer freien Alternative begonnen, dem *Open Graph Drawing Framework* (OGDF), dessen Funktionsumfang allerdings noch relativ beschränkt ist. Wenn die Implementierung von OGDF ausreichend weit vorangeschritten ist, wird es die AGD-Bibliothek in *Cupe* ablösen.

3.3.1 Allgemeine Zeichenmethoden

Mit *Cupe* ist es möglich beliebige Netzwerke zu erstellen; das hat natürlich zur Folge, dass es nicht möglich ist, die unterschiedlichen Kombinationen im voraus zu zeichnen. Deshalb wird ein metabolisches Netzwerk in *Cupe* zunächst ohne jedes Ausrichtung gezeichnet (siehe Abbildung 3.17 auf der nächsten Seite). Graphen, die nur wenige Reaktionen wiedergeben, können dann leicht mit kräftebasierten Verfahren ausrichtet (siehe Abbildung 3.18) werden. Für Pfade und größere Graphen eignen sich hierarchische (siehe Abbildung 3.20 auf Seite 57) oder planare (siehe Abbildung 3.19) Layout-Algorithmen. Die Layoutmethoden können über die Benutzeroberfläche von *Cupe* bequem aufgerufen werden. Über verschiedene Bedienelemente

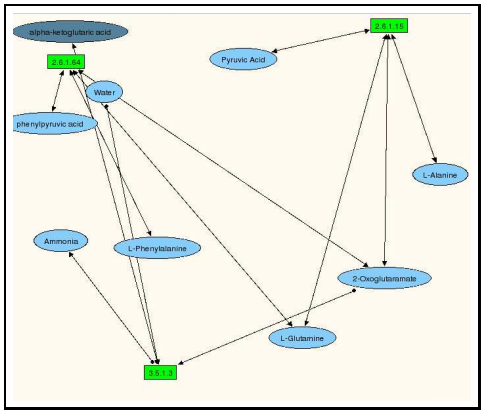


Abbildung 3.17: Zufälliges Layout Graphen von metabolischen Netzwerken, die automatisch erzeugt werden, werden zufällig und ohne jedes Layout auf der Zeichenfläche dargestellt.

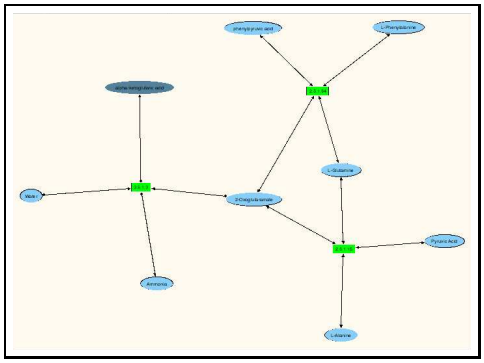


Abbildung 3.18: Kräftegerichtete Zeichung Cupe stellt alle kräftebasierten Layoutmethoden zur Verfügung, die von der Programmierbibliothek AGD angeboten werden.

können zusätzliche Optionen, wie etwa der Gitterabstand und die Animationsgeschwindigkeit, kontrolliert werden. Für jeden Editor kann die Zeichung des Graphen separat berechnet und auch wieder rückgängig gemacht werden.

3.3.2 Graphen metabolischer Netzwerke

Die grafische Wiedergabe von Biosyntheseprozessen beansprucht meist eine große Zeichenfläche. Wie gewöhnlich beginnt die Darstellung mittels *Cupe* mit einem zufällig ausgerichteten, unlesbaren Graphen des Reaktionsnetzwerks (siehe Abbildung 3.21 auf der nächsten Seite). Die planare Ausrichtung dieses Graphen führt zwar zu einer annähernd optimalen Verteilung der Knoten und zur weitgehenden Vermeidung von Kantenüberkreuzungen, bleibt aber dennoch schwer lesbar. Dieses wird zum einen durch die Menge der Knoten und ihrer zum Teil hohen Vernetzung und zum anderem durch die Länge der Kanten verursacht (siehe Abbildung 3.22 auf Seite 58).

Verringerung der Kantenlänge

Die schlechte Lesbarkeit des planar ausgerichteten Graphen der Glycolyse wird insbesondere durch die hohen Knotengrade sogenannter Nebenmetabolite oder auch Poolmetabolite verursacht (siehe Abschnitt 1.3.1 auf Seite 20). Diese Metabolite nehmen an zahlreichen Reaktionen teil und führen so im gezeichneten Netzwerk zu Kanten, die über den gesamten Graphen verlaufen. Um diese Kanten zu vermeiden, können solche Metabolite *gepoolt* werden (siehe Abbildung 3.23 auf Seite 58).
Hierzu kann über eine Schaltfläche das *Poollevel* von 0 an aufwärts erhöht werden. Welche Metabolite in der jeweiligen Stufe *gepoolt* werden, wird dynamisch beim Ein-

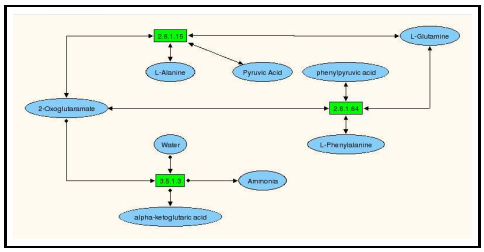


Abbildung 3.19: Planarisierung mit dem *Mixed-Model* Algorithmus Der Graph eines metabolischen Netzwerkes lässt sich gut mit den Planarisierungsmethoden aus AGD darstellen. Für kleine Graphen bietet sich hierzu besonders das Verfahren *Mixed-Model* an.

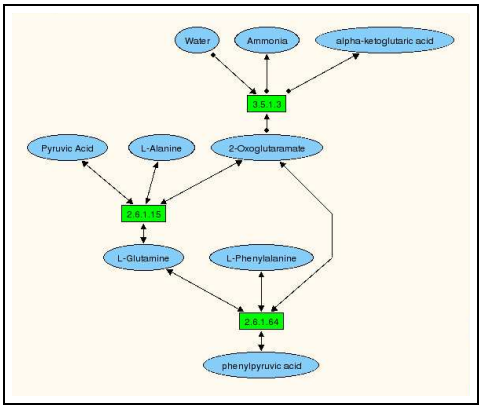


Abbildung 3.20: Hierarchische Darstellung Die hierarchische Darstellung metabolischer Netzwerke eignet sich besonders, um Pfade von Substrat zu Produkt wiederzugeben.

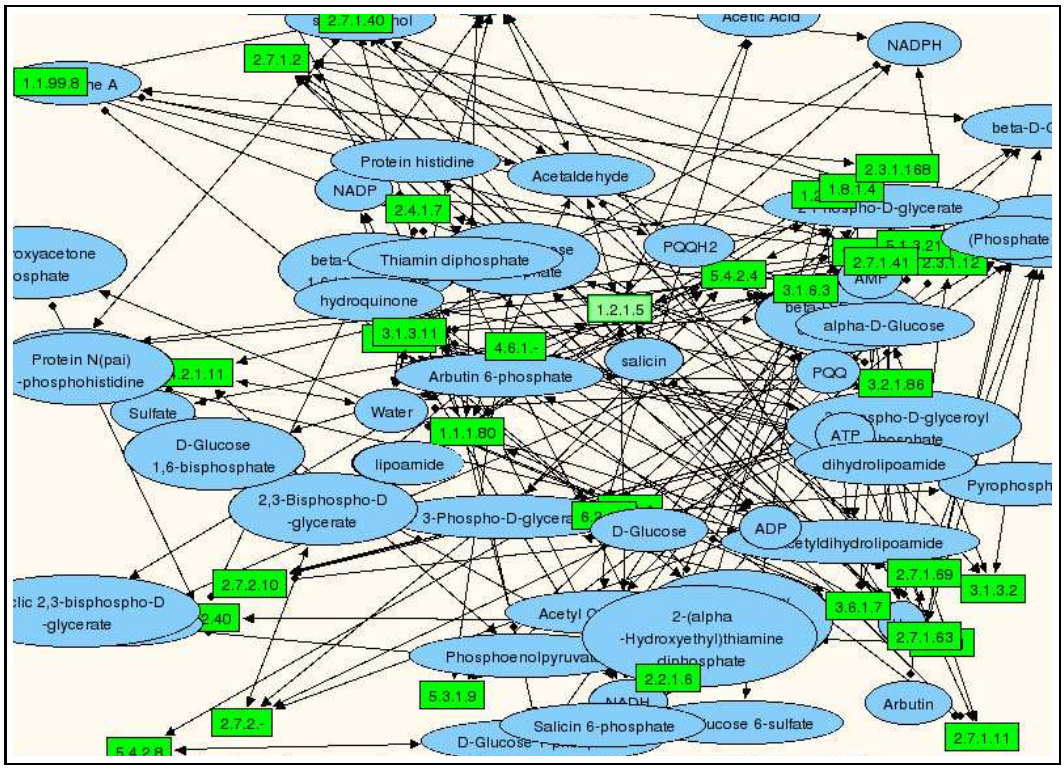


Abbildung 3.21: Glycolyse ohne Layout Initial ist jedes automatisch generierte Reaktionsnetzwerk ohne Layout. Dieses Netzwerk besteht aus 90 Molekülen und 163 Kanten. Die Kanten haben zusammen eine Länge von 47.822 Punkten. Der Graph ist unlesbar.

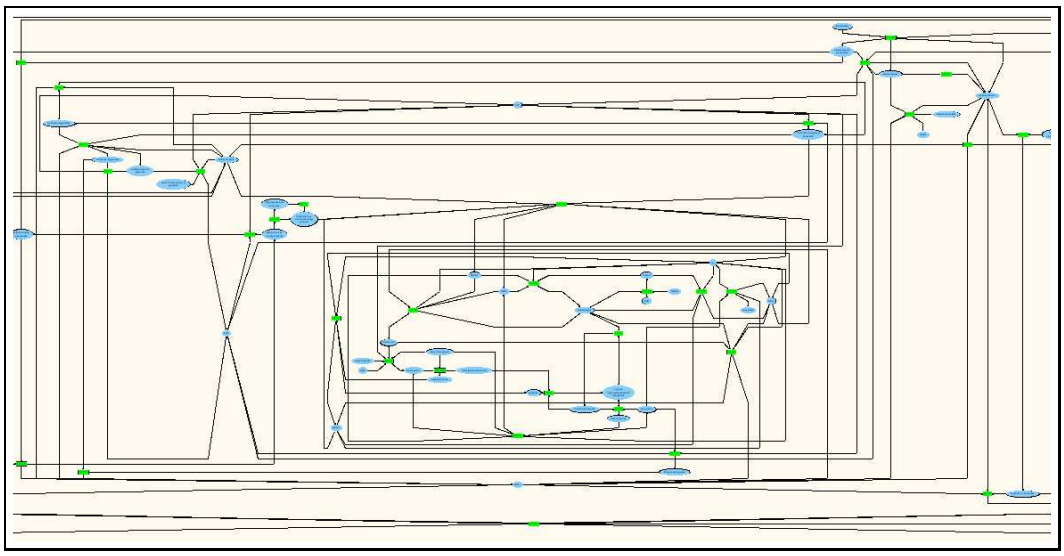


Abbildung 3.22: Glycolyse planar ausgerichtet Durch die planare Formatierung des Graphen werden die Moleküle und Kanten nahezu überschneidungsfrei ausgerichtet. Aber die Kantenlänge hat sich um den Faktor 4,7 verlängert. Die Kanten haben zusammen eine Länge von 224.488 Punkten. Der Graph ist weiterhin schlecht lesbar.

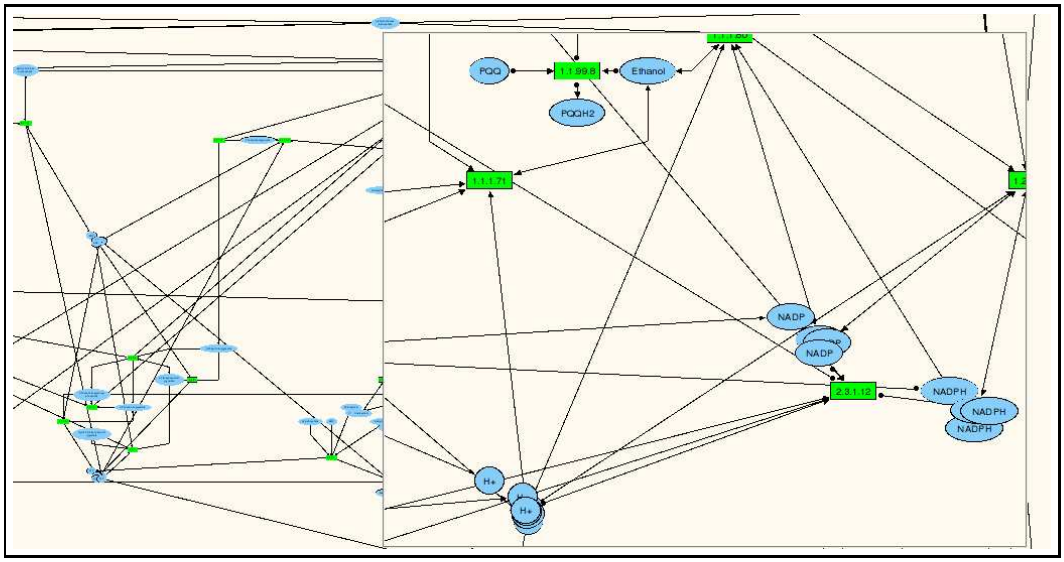


Abbildung 3.23: Pooling von Nebenmetaboliten Nebenmetabolite sind im Graphen durch einen hohen Knotengrad gekennzeichnet (vergleiche Abbildung 3.22). Diese Moleküle können automatisch oder manuell gepoolt werden. Dadurch wird der Knotengrad jedes Nebenmetaboliten auf 1 gesetzt, die Anzahl der Knoten wird erhöht, aber die Anzahl der Kanten bleibt mit 163 konstant.

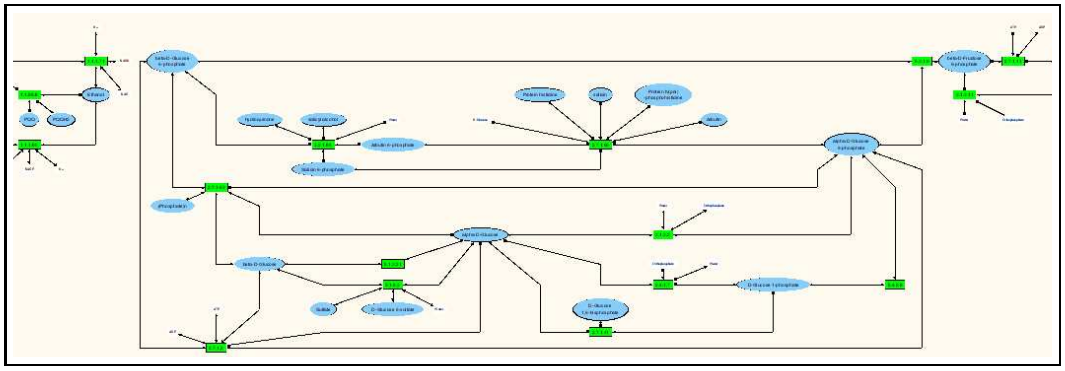


Abbildung 3.24: Glycolyse gepoolt und planar ausgerichtet Gepoolte Metabolite können automatisch oder manuell ausgewählt und einheitlich formatiert werden. Die Lesbarkeit des metabolischen Graphen hat sich deutlich verbessert (vergleiche Abbildung 3.22 auf der vorherigen Seite), obwohl 55 neue Knoten eingefügt wurden. Die Kantenlänge hat sich um den Faktor 2.3 auf 98.978 Punkte verkürzt.

---

lesen der Reaktionsdatenbank zum Programmstart berechnet. Für jede Substanz wird dabei bestimmt, an wie vielen Reaktionen sie teilnimmt. Die Moleküle, die an den meisten Reaktionen teilnehmen, werden in *Cupe* auch zuerst gepoolt. Dieses Verfahren hat den Vorteil, dass es sich automatisch mit dem Datenbestand synchronisiert und auch für einzelne Metabolite direkt angewendet werden kann.

Durch das *Pooling* der hochvernetzten Metabolite werden dem Graphen zwar neue Knoten hinzugefügt, die Anzahl der Kanten bleibt aber konstant. Nach der Neuberechnung des Graphenlayouts hat sich die Länge aller Kanten zusammen um den Faktor 2,3 verringert, insgesamt hat sich die Lesbarkeit trotz der 55 neu eingefügten Knoten deutlich verbessert (siehe Abbildung 3.24). Über eine weitere Schaltfläche können alle Nebenmetabolite zugleich ausgewählt und einheitlich formatiert werden. Dadurch kann die Übersichtlichkeit des metabolischen Netzwerkes zusätzlich erhöht werden.

Die Implementierung der Cluster als Superknoten (siehe Abschnitt 3.2.1) ermöglicht es, den Graphen noch weiter zu vereinfachen, ohne dass dabei auf Informationen verzichtet werden müsste. Für diese Operation muss man lediglich alle Knoten mit einem Knotengrad gleich 1 direkt mit dem Enzym der Reaktion zusammenfassen. Ein Knotengrad von 1 bezeichnet dabei, dass von diesem Metaboliten nur eine Kante ausgeht. Im Regelfall, wenn der Graph bipartit ist (1.3.3), führt diese Kante dann zu einem Enzym. Dadurch, dass man die Zeichenfläche eines Clusters kollabieren kann, kann man diese Kanten und Knoten in einen Knoten zusammenfassen und je nach Bedarf ein- und ausblenden (siehe Abbildung 3.25 auf der nächsten Seite). Für dieses Verfahren wurde ein kleines Plugin geschrieben, das unter dem Menüeintrag *Cluster Algorithms* abgerufen werden kann (siehe 3.14 auf Seite 51, rechts unten).

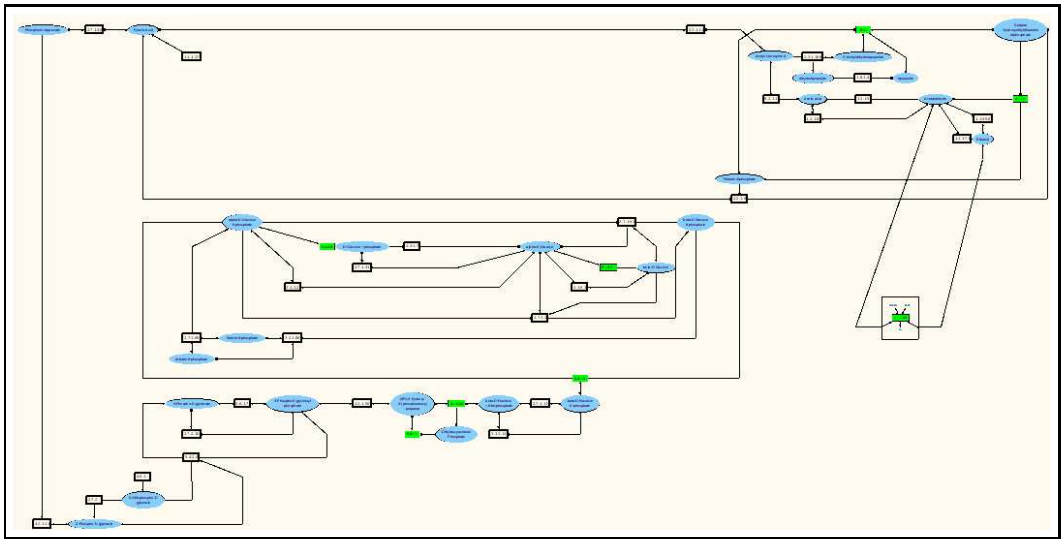


Abbildung 3.25: Clustern endständiger Metabolite Alle Metabolite, die einen Knotengrad gleich 1 haben, können in einem Enzym-Cluster zusammengefasst und dieser dann kollabiert werden. Diese Molekülcluster (siehe Abbildung 3.16 auf Seite 54) können wieder expandiert werden, um sich die entsprechenden Subnetzwerke anzusehen. In der Abbildung wird rechts in der Mitte des Graphen ein expandierter Enzym-Cluster gezeigt. Insgesamt verringert sich die Anzahl der Knoten auf 62 und die Anzahl der Kanten auf 85. Nach der planaren Ausrichtung beträgt die Kantenlänge nur noch 31.190 Punkte.

Tabelle 3.1: Verbesserung der Lesbarkeit

Der anfangs nicht ausgerichtete Graph der Glykolyse hat hat zwar eine insgesamt geringe Kantenlänge, ist aber unlesbar (3.21). Über mehrere Schritte (vergleiche 3.22 bis 3.24) kann die Lesbarkeit des Netzwerkes zunehmend verbessert werden. Insgesamt kann die Anzahl der Knoten um 31%, die Anzahl der Kanten um 48% und die Länge der Kanten um 34% reduziert werden.

Layout	Knoten	Kanten	Kantenlänge
Initial	90	163	47822 Punkte
Planar	90	163	224488 Punkte
Gepoolt + Planar	145	163	98978 Punkte
Gepoolt + Geclustert + Planar	62	85	31190 Punkte

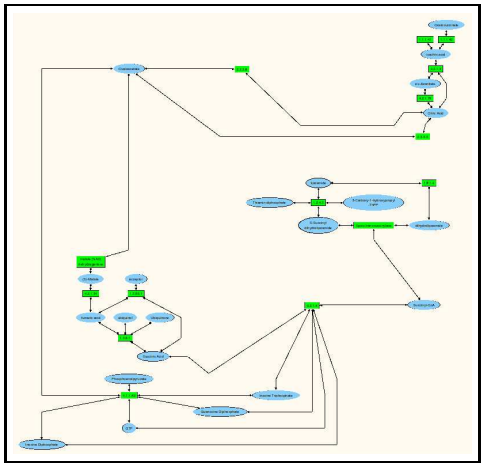


Abbildung 3.26: Citratzyklus von *Arabidopsis thaliana* Dieser Graph enthält bereits Molekülcluster, die unabhängig vom Gesamtgraphen formatiert werden können. Auf diese Weise können bestimmte Teile eines Graphens auch von Veränderungen ausgeschlossen werden (siehe „Mentale Karte“ in 1.1.2 auf Seite 4).

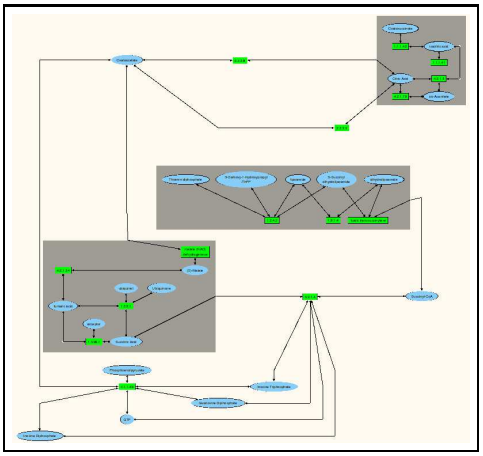


Abbildung 3.27: Layout bestimmter Subgraphen Die grau hinterlegten Subgraphen sind in Molekülclustern abgelegt, die unabhängig vom Gesamtgraphen ausgerichtet wurden.

3.3.3 Partielles Layout

Algorithmen zum Zeichnen von Graphen müssen viele Eigenschaften gleichzeitig optimieren. Die exakte Berechnung, wie Knoten am besten platziert werden können, welcher Kantenverlauf die wenigsten Überkreuzungen erzeugt oder auch welche Symmetrien in einer Zeichnung auftauchen können, benötigt eine exponentielle Laufzeit. Deswegen versucht man die Berechnung durch verallgemeinernde Annahmen zu vereinfachen und sich einer optimalen Lösung lediglich anzunähern. Bedingt durch dieses heuristische Vorgehen, kann die erzeugte Zeichnung eines Graphen bei jeder Neuberechnung unterschiedlich sein. Aber auch wenn man den Graphen bearbeitet und neue Elemente hinzufügt, wird bei einer neuen Berechnung der Zeichung der Graph meistens völlig umgestellt. Der Anwender muss sich also erstmal wieder neu orientieren und sich an die neue Sichtweise gewöhnen. Diese „Unstetigkeit“ ist besonders dann unerwünscht, wenn der Anwender das Layout eines Graphen manuell verändert hat. Um zu vermeiden, dass diese manuellen Veränderungen bei dem nächsten Layoutvorgang verloren geht, kann man Teile eines metabolischen Netzwerkes in Cluster überführen. Diese Cluster können separat ausgerichtet werden und verändern die Zeichung des Subgraphen in ihnen nicht, wenn der Gesamtgraph neu gezeichnet wird. Diese Technologie kann auch rekursiv angewendet werden und bildet die Grundlage für die automatische Clusterung und die Implementation einer *Mentalen Karte* (siehe Abschnitt 1.1.2 auf Seite 4).

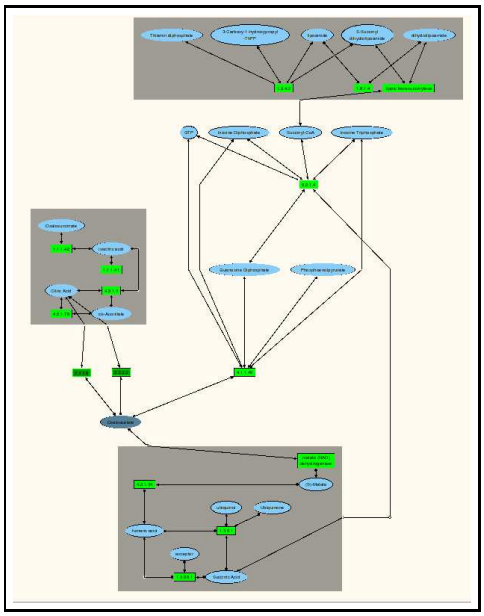


Abbildung 3.28: Neues Graphlayout Der Gesamtgraph ist neu ausgerichtet worden, ohne dass die Cluster ihr Layout veränderten.

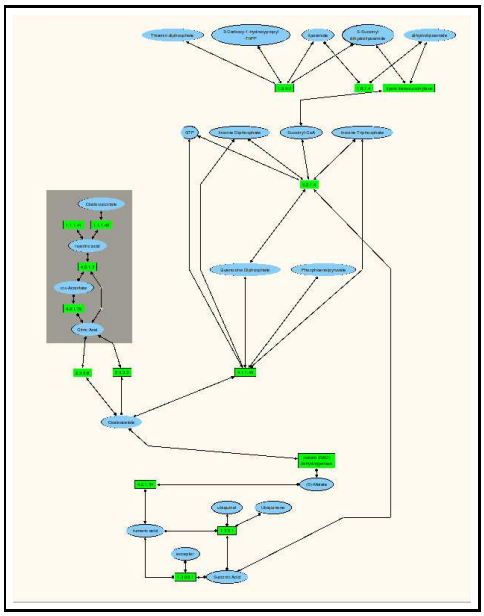


Abbildung 3.29: Layout eines Clusters Um die Kantenkreuzungen im grau hinterlegten Cluster zu vermeiden, wurde die Zeichung des Graphen in diesen Cluster neu berechnet.

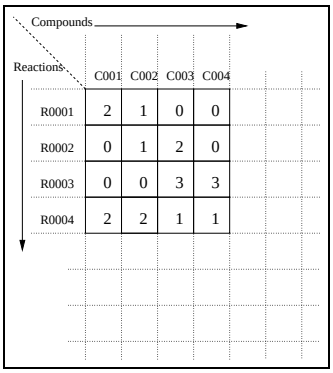


Abbildung 3.30: Reaktionsmatrix Eine 0 symbolisiert, dass die Verbindung (Compound) nicht an der Reaktion teilnimmt. 1, 2 und 3 kennzeichnen die Art der Teilnahme (Substrat, Produkt, beides).

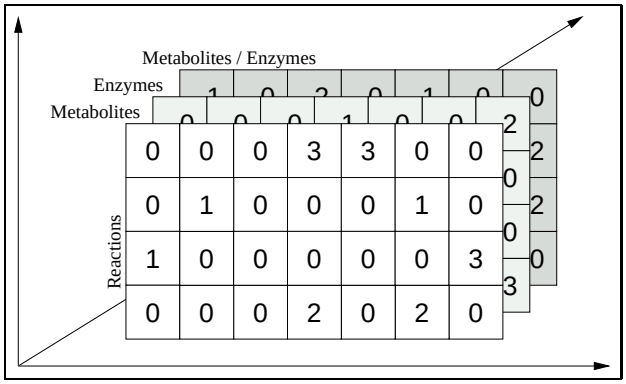


Abbildung 3.31: Cubic-Matrix Die dreidimensionale Cubic-Matrix setzt sich aus mehreren Reaktionsmatrizen (siehe links) zusammen. Jede dieser Matrizen ist in ihren zwei Dimensionen dynamisch erweiterbar, und die Cubic-Matrix kann ebenfalls um neue Matrizen erweitert werden. Gegenwärtig enthält die Cubic-Matrix die Matrizen: Metabolite × Reaktionen, Enzyme × Reaktionen und Metabolite × Enzyme.

Die Abbildungen 3.26 bis 3.29 zeigen, wie in *Cupe* eine Stoffwechselkarte des Citratzyklus' neu ausgerichtet wird, ohne dass sich die fixierten Subgraphen ebenfalls neu ausrichten. Ebenso ist es möglich, nur für einen Cluster eine neue Zeichung zu berechnen und den Rest des Graphen unverändert zu lassen.

3.4 Cubic-Matrix

Um die Daten über die Stoffwechselreaktionen für die Analyse und Darstellung in *Cupe* bereitzustellen, werden die einzelnen Reaktionen in Reaktionsmatrizen eingetragen (siehe Abbildung 3.30). Eine einfache Matrix, die die Verknüpfung zwischen Metaboliten und Enzymen wiedergibt, ist jedoch nicht ausreichend, um auch einen Pfad von Substrat zu Produkt anzugeben. Aus diesem Grund wird zusätzlich eine Matrix erzeugt, die die Teilnahme von Enzymen an den Reaktionen wiedergibt. Eine weitere Matrix speichert die gleichen Informationen für die Metabolite. Alle drei Matrizen werden zu einer dreidimensionalen *Cubic-Matrix* zusammengefasst.

3.4.1 Cubic-Sparse-Matrix

Die direkte Implementierung der internen Datendarstellung mittels der neu konzipierten, dynamischen *Cubic-Matrix* führt zu einem immensen Speicherbedarf. Eine einzelne Matrix mit 40.000 Verbindungen und 30.000 Reaktionen benötigt beispielsweise bereits ca. 4,5 GB Speicher. Wenn man bedenkt, dass zur jeder Reaktion mehrere unterschiedliche Arten der Information gespeichert und dass gleichzeitig mehrere solcher Matrizen miteinander verglichen werden sollen, wird schnell klar,

dass eine naive Implementierung dieser Matrizen nicht im Speicher gehalten werden kann.
Da die einzelnen Verbindungen im Durchschnitt jedoch nur an zwei bis vier Reaktionen teilnehmen, ist der weitaus größte Anteil der Matrixzellen mit dem Wert 0 besetzt. Für derart gering besetzte Matrizen eignen sich „Sparse-Matrix-Datentypen“ (siehe Abbildung 3.32). Für *Cupe* ist aus diesem Grund eine eigene Sparse-Matrix-Klasse entwickelt wurden, die die folgenden Anforderungen erfüllt.

- Die Implementierung muss ISO-C++-konform sein, damit die Algorithmen der Standard Template Bibliothek benutzt werden können (vergleiche Stroustrup, 2000).
- Um die Datentypen, die mit einer Matrix assoziiert werden können, nicht zu beschränken, soll die Sparse-Matrix-Klasse als Template-Klasse implementiert werden.
- Als „Sparse-Wert“ sollen auch Werte ungleich von Null möglich sein.
- Es sollen ISO-C++ konforme Zellen-, Spalten- und Zeilen-Iteratoren angeboten werden.
- Die Matrix soll sich in allen Dimensionen dynamisch verhalten.
- Zusätzlich zu den Standardalgorithmen soll spalten- und zeilenweises Löschen möglich sein.
- Der lesende Zugriff sollte sehr schnell möglich sei.
- Das Benutzerinterface dieser Klasse soll die Schnittstelle zu – auch mehrdimensionalen – C-Arrays vollständig anbieten.

Die Kombination von dynamischem Speichermodell und hoher Geschwindigkeit konnte durch eine Vereinigung von Hash-Tabellen und assoziativen Listen erreicht werden. Dadurch ergibt sich die Notwendigkeit, eine Hashfunktion anzubieten, wenn man die dynamische Sparse-Matrix-Klasse benutzen möchte.
Die Tatsache, dass diese neue Sparse-Matrix auf die gleiche Art benutzt werden kann wie einfache C-Arrays, ermöglicht es, sie leicht in anderen Projekten zu nutzen. Mitunter muss hierzu lediglich eine Deklaration geändert werden. So war es auch einfach, ihre Leistungsfähigkeit zu testen. Im direkten Vergleich mit einer einfachen assoziativen Matrix auf der Grundlage der STL (siehe Abschnitt 2.5.1 auf Seite 37) ist die *Cubic-Sparse-Matrix* um zwei Größenordnungen schneller. Verglichen mit der statischen Sparse-Matrix von Boost (siehe Abschnitt 2.5.5 auf Seite 38), ist die *Cubic-Sparse-Matrix* immer noch um eine Größenordnung schneller. Lediglich im direkten Vergleich mit einem zweidimensionalen C-Feld vom Datentyp Integer ist die *Cubic-Sparse-Matrix* um den Faktor 1,7 langsamer.

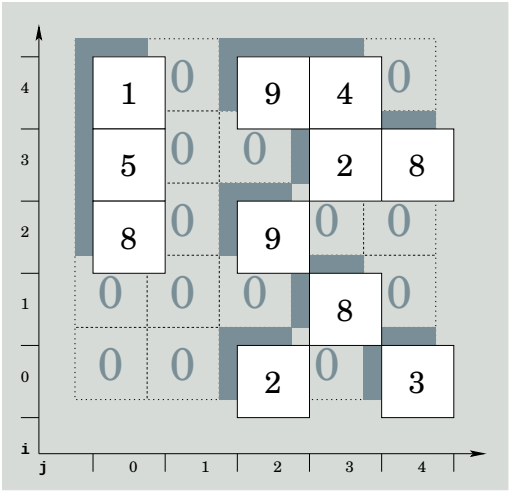


Abbildung 3.32: Sparse-Matrix. Der „Sparse-Wert“ dieser Matrix ist 0. Alle Zellen mit einem Wert ungleich 0 sind weiß dargestellt. Alle Zellen mit einem Wert gleich 0 sind als graue Schatten dargestellt. Nur weiße Zellen werden tatsächlich gespeichert. Wenn ein Index $m(i, j)$ eine tatsächlich gespeicherte Zelle adressiert, wird diese Zelle geliefert, sonst eine neue, nicht gespeicherte Zelle mit den Wert 0. Wird einer neuen Zelle ein Wert ungleich 0 zugewiesen, wird sie gespeichert. Wird einer gespeicherten Zelle ein Wert gleich 0 zugewiesen, wird sie gelöscht.

Die Form der internen Repräsentation von Reaktionsnetzwerken mittels einer Sparse-Martix, worde zusammen mit den Entwicklern des *Pathway Hunter Tool* entwurfen (Rahman S. A., 2004). Die Implementation erfolgte für *Cupe* in Form einer unabhängigen Template-Bibliothek und wird sehr erfolgreich und unverändert von dem Programm *Pathway Hunter Tool* mitverwendet.
Die vollständig dokumentierte Dynamische Sparse-Matrix-Template-Klasse ist unter der GNU Public License erhältlich.

3.5 Mengenoperationen für metabolische Netzwerke

Für die vergleichende Analyse von metabolischen (Sub-)Graphen wurde eine Programmierschnittstelle auf der Grundlage der Mengenlehre implementiert. Für diesen Zweck musste die *Cubic-Sparse-Matrix* um ein generisches Interface für die Bildung der Differenz[1]-, Schnitt- und Vereinigungsmenge erweitert werden (siehe 3.5.1). Um Mengenfunktionen auf Reaktionen oder Pfade anzuwenden, musste eine Syntax und Semantik zur logischen Beschreibung von Stoffwechselreaktionen entwickelt werden (siehe 3.5.2 auf der nächsten Seite). Die Bezeichner dieser Qualifizierung orientieren sich an den Begriffen der Thermodynamik, die Semantik ihrer Mengenoperationen ist neu entwickelt worden (siehe 3.5.3 auf Seite 68).

3.5.1 Erweiterte Cubic-Sparse-Matrix

Die Erweiterungen bestehen in der Implementation der Operatoren OR, AND und XOR für alle Kombinationen von Matrizen, Spalten und Zeilen. Besonderes Augenmerk musste auf die Bestimmung der besetzten Zellen von zueinander in Beziehung gesetzten Spalten und Zeilen gerichtet werden, da die *Cubic-Sparse-Matrix* als dy-

[1] gemeint ist die symmetrische Differenz

Tabelle 3.2: Thermodynamische Qualifizierung von Metaboliten Die thermodynamische Qualifizierung eines Metaboliten stellt die Beschreibung des Gradienten der Metabolitkonzentration während der Reaktion dar.

Bezeichner	Kurzform	Bedeutung
POSITIVE	P	Die Metabolitkonzentration nimmt ab
NEGATIVE	N	Die Metabolitkonzentration nimmt zu
BALANCED	B	Die Netto-Metabolitkonzentration verändert sich nicht
UNKNOWN	U	Die Konzentrationsänderung ist nicht bekannt
NONE	-	Der Metabolit nimmt an der Reaktion nicht teil

Tabelle 3.3: Thermodynamische Qualifizierung von Enzymen Die thermodynamische Qualifizierung eines Enzyms stellt die Beschreibung der Reaktionsrichtung einer von diesem Enzym katalysierten Reaktion dar. Für den Spezialfall, dass eine Reaktion ohne katalysierendes Enzym abläuft, wird ein „leerer“ Enzymbezeichner eingeführt.

Bezeichner	Kurzform	Bedeutung
POSITIVE	P	Katalyse der Hinreaktion
NEGATIVE	N	Katalyse der Rückreaktion
BALANCED	B	Katalyse von Hin- und Rückreaktion
UNKNOWN	U	Der Einfluss auf die Reaktion ist nicht bekannt
NONE	-	Das Enzym katalysiert die Reaktion nicht

namische Datenstruktur und ohne Indexgrenzen entworfen wurde. Für diesen Zweck wurde ein speziell angepasster *Merge*-Algorithmus entwickelt.

3.5.2 Thermodynamisch orientierte Beschreibung von Reaktionen

Der Einfluss jedes Reaktionsteilnehmers kann über eine quintäre Logik beschrieben werden. Diese quintäre Logik wird als thermodynamisch orientierte Qualifizierung eingeführt. Die Grundelemente dieser Logik sind die Bezeichner: `POSITIVE`, `NEGATIVE` und `BALANCED`.

Katalysierte Reaktionen

Die thermodynamisch orientierte Qualifizierung bezieht sich darauf, dass sich die Konzentration eines Metaboliten während einer Reaktion in thermodynamisch definierter Weise verändert (siehe Tabelle 3.2). Ähnlich zu dem Konzentrationsgradienten eines Metaboliten, gibt es einen Gradienten als Reaktionsrichtung. Dieser Gradient *über* einer Reaktion kann mit den gleichen Bezeichnern beschrieben werden (siehe Tabelle 3.3).

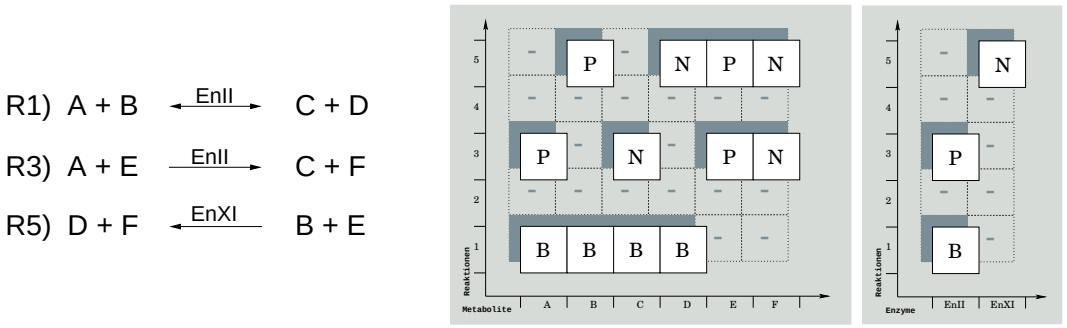


Abbildung 3.33: Encodieren von Reaktionen Für alle an den Reaktionen beteiligten Molekülen (**links**) wird der Einfluss auf die Reaktion bestimmt und in die CUBIC-Matrix eingetragen (vergleiche die Tabellen 3.2 und 3.3). Die CUBIC-Matrix ist zweigeteilt (**rechts**). Zum einen, weil so der Datenbestand zur Laufzeit leicht erweitert werden kann, zum anderen, um so Enzyme und Metabolite getrennt zu behandeln.

Der Sparse-Wert

Die Vielzahl aller Moleküle nimmt nur an sehr wenigen Reaktionen teil. Um auszudrücken, dass ein Molekül an einer Reaktion nicht beteiligt ist wird als vierter Bezeichner `NONE` eingeführt. `NONE` stellt gleichzeitig den „Sparse-Wert“ der CUBIC-Matrix dar. Erst diese besondere Bedeutung von `NONE` ermöglicht es, alle Moleküle (M) und alle Reaktionen (R) in einer Sparse-Matrix ($S = M \times R$) zu speichern (siehe 3.5.1 auf Seite 65).

Unbekannte Interaktionen

Die Tatsache, dass in einer Sparse-Matrix der „Sparse-Wert“ nicht gespeichert wird, macht es erforderlich, einen weiteren Bezeichner `UNKNOWN` einzuführen. Mit `UNKNOWN` können Moleküle in die Matrix eingetragen werden, von denen man noch nicht weiß, auf welche Weise sie an Reaktionen teilnehmen.

Nicht katalysierte Reaktionen

Eine Besonderheit stellen die Reaktionen dar, die ohne Enzyme ablaufen. Innerhalb der CUBIC-Matrix kann die ($S = M \times E$)-Beziehung nur aufrecht erhalten werden, indem man vorerst „leere“ Enzyme als Katalysator einträgt. Eine endgültige Lösung ist noch nicht definiert.

Eintragen von Reaktionen in die CUBIC-Matrix

Zur Veranschaulichung werden in Abbildung 3.33 drei Beispiel-Reaktionen in eine CUBIC-Matrix eingetragen. Diese Eintragungen erfolgen auf der Grundlage der zuvor im Abschnitt 3.5.2 eingeführten thermodynamisch orientierten Qualifizierung.

Tabelle 3.4: OR-Operation für thermodynamische Qualifizierer
Die OR-Operation entspricht der Vereinigung zweier Mengen ($A \bigcup B$).

OR	POSITIVE	NEGATIVE	BALANCED	NONE	UNKNOWN
POSITIVE	P	B	B	P	P
NEGATIVE	B	N	B	N	N
BALANCED	B	B	B	B	B
NONE	P	N	B	-	-
UNKNOWN	P	N	B	-	U

Tabelle 3.5: XOR-Operation für thermodynamische Qualifizierer
Die XOR-Operation entspricht der symmetrische Differenz zweier Mengen ($A \bigoplus B$)

XOR	POSITIVE	NEGATIVE	BALANCED	NONE	UNKNOWN
POSITIVE	-	B	N	P	P
NEGATIVE	B	-	P	N	N
BALANCED	N	P	-	B	B
NONE	P	N	B	-	-
UNKNOWN	P	N	B	-	U

Beispiele für die Bedeutung von `UNKNOWN` findet man in den Abbildungen 3.35 bis 3.37 auf Seiten 70–71.

3.5.3 Mengenoperationen für thermodynamische Qualifizierer

Die thermodynamischen Bezeichner von Metaboliten und Enzymen können untereinander über Mengenoperationen in Beziehung gesetzt werden. Die Bedeutung für die Grundbezeichner `POSITIVE`, `NEGATIVE` und `BALANCED` kann man sich leicht vorstellen, wenn man sie in Reaktionspfeile übersetzt: $\rightarrow$, $\leftarrow$ und $\leftrightarrow$. Eine OR-Verknüpfung von `POSITIVE` und `NEGATIVE` wird so zu $\rightarrow OR \leftarrow$. Die Überlagerung der Reaktionspfeile führt zu $\leftrightarrow$ = `BALANCED`. Diese Bild ist allerdings nur noch für `NONE` (—), nicht aber für `UNKNOWN` übertragbar. Die Tabellen 3.4 bis 3.6 auf dieser Seite listen alle möglichen logischen Verknüpfungen auf.

Tabelle 3.6: AND-Operation für thermodynamische Qualifizierer
Die AND-Operation entspricht der Schnittmenge zweier Mengen ($A \subset B$)

AND	POSITIVE	NEGATIVE	BALANCED	UNKNOWN	NONE
POSITIVE	P	-	P	-	-
NEGATIVE	-	N	N	-	-
BALANCED	P	N	B	-	-
UNKNOWN	-	-	-	U	-
NONE	-	-	-	-	-

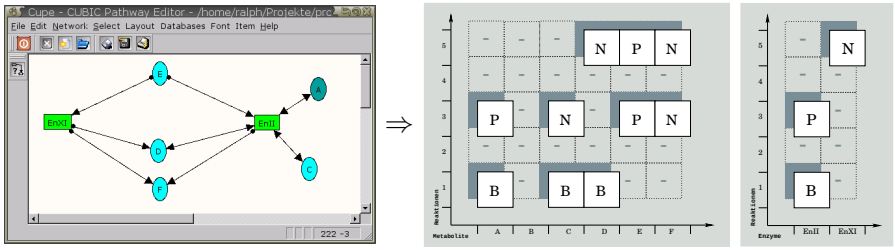


Abbildung 3.34: Editor-Graph *versus* Editor-Matrix Zu jedem im Editor gezeichneten Graph wird eine Editor-Matrix erstellt. **links:** Gezeichneter Graph; **rechts:** Korrespondierende Editor-Matrix

3.5.4 Anwendung der Mengenoperationen

Mit Hilfe der neu definierten Mengenfunktionen für Stoffwechselreaktionen ist es möglich, folgende Fragestellungen zu untersuchen:

- Wie werden neu eingefügte Moleküle in einen bestehenden Graph integriert (siehe 3.5.5)?
- Welche Vernetzung besteht für Graphen untereinander?
- Wie unterscheiden sich die Graphen von Stoffwechselwegen oder unterschiedlichen Organismen (Abbildung 3.40 auf Seite 73)?
- Welches sind die Gemeinsamkeiten verschiedener Graphen (Abbildung 3.41 auf Seite 74)?
- Welche der Teile des Netzwerkes stellen unbekannte Reaktionen dar?

Bis auf den letzten Punkt können in *Cupe* alle diese Fragen beantwortet werden. Die Letzte läßt sich auch beantworten, aber es ist noch keine interaktive, graphische Benutzerschnittstelle entwickelt worden.

3.5.5 Schritt-für-Schritt-Beispiel

Die schrittweise Anwendung der Mengenoperationen wird am Beispiel der Integration von Molekülen in einen bestehenden Graphen gezeigt. Für dieses Beispiel werden die bereits gezeigten Beispielreaktionen (siehe 3.5.2 auf Seite 67) verwendet.

0 - Editor-Matrix

Zuerst wird, für den im Editor gezeichneten Graphen, eine Editor-Matrix erstellt. Im Unterschied zur CUBIC-Matrix enthält diese Matrix nur die gezeichneten Reaktionen des Graphen. Hierbei stellt die Identifikation der Reaktionen die algorithmische Herausforderung dar (Abbildung 3.34).

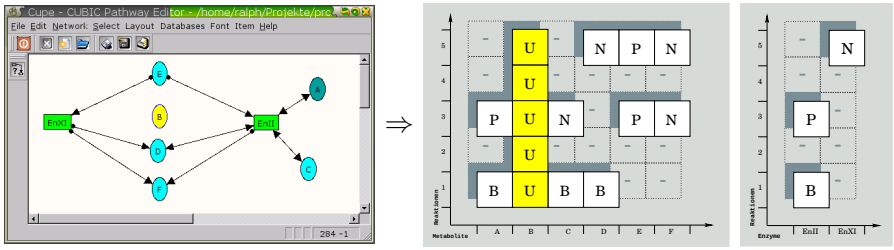


Abbildung 3.35: Erster Schritt zur Integration Ein neuer Metabolit wird eingeführt und selektiert. Eine Interaktion mit dem Reaktionsnetzwerk besteht nicht. Der erste Schritt ist daher, den ausgewählten Metaboliten mit unbekannter Interaktion in die Editor-Matrix einzufügen. Hier zeigt sich die Notwendigkeit, die Interaktion eines Moleküls mit dem Editor-Netzwerk als `UNKNOWN` in die korrespondierende Editor-Matrix eintragen zu können. `NONE` ist hierfür nicht geeignet, weil das dem „Sparse-Wert“ der CUBIC-Matrix entspricht und so die Information, welches Molekül neu eingefügt wurde, verloren ginge. **links:** Gezeichneter Graph; **rechts:** Korrespondierende Editor-Matrix

1 - Unbekannte Vernetzung

Das ausgewählte Molekül wird mit einer unbekannten Vernetzung in die Editor-Matrix eingetragen (Abbildung 3.35).

2 - Bekannte Reaktionen

Die ausgewählten Moleküle bilden die Anfrage-Matrix[2]. Diese Anfrage-Matrix wird mit der CUBIC-Matrix vereinigt (nach Tabelle 3.4), so dass als Ergebnis alle bekannten Reaktionen für die Moleküle der Anfrage-Matrix in diese eingetragen werden (vergleiche Abbildung 3.36 auf der nächsten Seite)

3 - Unterschiede zur Editor-Matrix

Zwischen der Anfrage-Matrix und der Editor-Matrix wird die Symmetrische Differenz (nach Tabelle 3.5) gebildet. Das Ergebnis ist eine Differenz-Matrix, in der alle noch nicht eingetragenen Reaktionen abgebildet sind. Diese Matrix kann dann in die Editor-Matrix eingetragen werden. Die beiden letzten Schritte können zugleich durch die Vereinigung von Anfrage- und Editor-Matrix durchgeführt werden (Abbildung 3.37 auf der nächsten Seite). Dabei geht jedoch die Information über die Unterschiede zwischen beiden Matrizen verloren.

4 - Neue Editor-Matrix

Die neue Editor-Matrix bzw. die Differenz-Matrix wird in den Graphen des Stoffwechselsystems eingetragen (Abbildung 3.38 auf Seite 72). An dieser Stelle können

[2]Für ein Molekül eine Spalte bzw. Vektor

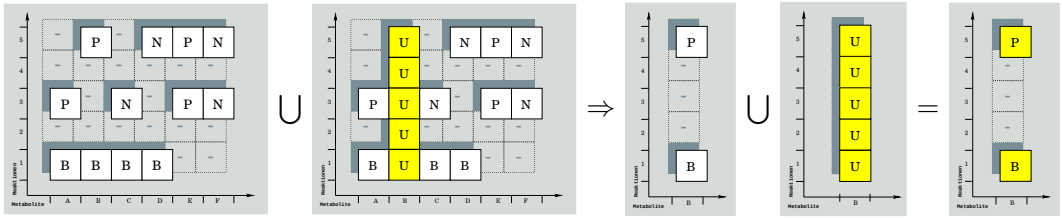


Abbildung 3.36: Ermittlung der bekannten Reaktionen Von links nach rechts: CUBIC-Matrix mit allen Reaktionen; Editor-Matrix mit den gezeichneten Reaktionen; Alle Reaktionen die für B bekannt sind; Unbekannte Reaktionen für B als Anfrage-Matrix; Neue Reaktionen in der Anfrage-Matrix. Die, als umfassende Datenstruktur konzipierte, *CUBIC-Matrix* enthält natürlich sehr viel mehr Reaktionen, als im Editor gezeichnet. Die Vereinigung von *CUBIC-Matrix* und Editor-Matrix führe somit zur Abbildung aller Reaktion in die Editor-Matrix. Hier sind aber nur die Reaktionen von Interesse, an denen die ausgewählten Moleküle teilnehmen. Daher werden nur die Spalten aus der *CUBIC-Matrix*, die mit den angefragten Molekülen übereinstimmen, mit der Anfrage-Matrix vereinigt. Die Auswahl der richtigen Spalten und die Bestimmung der Indexgrenzen, wird nativ über die Spalten-Iteratoren der *CUBIC-Matrix* unterstützt (siehe 3.32 auf Seite 65). Die Vereinigung der Spalten wird über Tabelle 3.4 auf Seite 68 definiert. Als Ergebnis dieser Prozedur erhält man eine Anfrage-Matrix mit allen bekannten Reaktionen für die ausgewählten Moleküle (rechts).

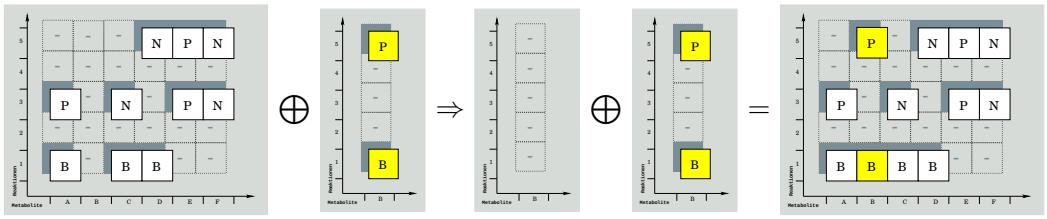


Abbildung 3.37: Bestimmung der neuen Kanten Von links nach rechts: Editor-Matrix; Anfrage-Matrix mit Reaktionen für B; Bereits gezeichnete Reaktionen (keine); Reaktionen für B; Editor-Matrix mit neuen Reaktionen für B. Zur Bestimmung der neuen Kanten wird die symmetrische Differenz (siehe Tabelle 3.5 auf Seite 68) zwischen Editor-Matrix und Anfrage-Matrix (Abbildung 3.36 rechts) gebildet. Durch diese XOR-Operation, werden auch bereits gezeichnete Teil-Reaktionen erkannt, und so mehrfache Einträge in den Graphen vermieden. Dieses Verhalten erleichtert z.B. die automatische Formatierung mittels Planarisierung. Aus den gleichen Gründen wie zuvor (Abbildung 3.36), wird hier wieder mit Spalten-Iteratoren gearbeitet.

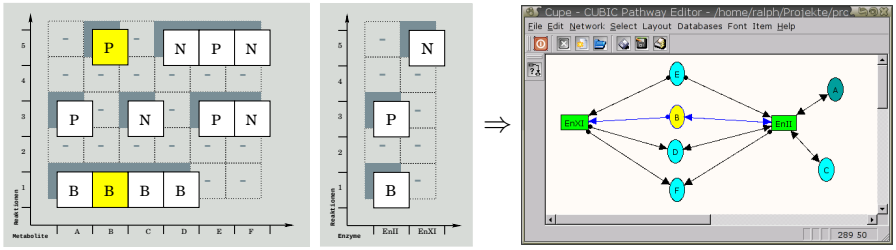


Abbildung 3.38: Editor-Matrix *versus* Editor-Graph (links) Neue Editor-Matrix; (rechts) Neue Kanten im gezeichnetem Graph. Der letzte Schritt bei der Bestimmung der Reaktionspartner ausgewählter Moleküle ist das Eintragen der neuen Kanten in den Graphen des metabolischen Netzwerkes.

Methoden zur Graphenformatierung angesetzt werden, der Knotengrad von „Nebenmetaboliten“ auf 1 gesetzt werden, usw.

Skizzierung weiterer Anwendungen

Die folgenden Abbildungen (siehe Abbildung 3.39 bis 3.41) skizzieren weitere Anwendungen und Fragestellungen, die mit den Mengenoperationen auf den Stoffwechsel-Matrizen möglich sind.

3.6 Analyse-Methoden

Für die Analyse metabolischer Netzwerke sind sehr viele Ansätze denkbar. Das Spektrum der möglichen Methoden erstreckt sich von der einfachen Darstellung von Pfadanalysen (PHT Rahman S. A., 2004) bis hin zur animierten Simulation mit eigenen Konfigurationsdialogen (z.B. Hartman, K., 2005). Viele dieser Methoden werden über die *Cupe*-eigene Pluginschnittstelle bereitgestellt (ab 3.7 auf Seite 78). *Cupe* verfügt zudem aber auch über eine leistungsstarke Datenbankschnittstelle, welche es erlaubt, Reaktionsnetzwerke gemäß den eigenen Bedürfnissen zu konstruieren und zu vergleichen.

3.6.1 Reaktionsdatenbank

Viele der frei zugreifbaren Stoffwechseldatenbanken und die Brendadatenbank (Kanehisa & Goto, 1999; Schomburg *et al.*, 2000, ...) sind am Cologne University BioInformatics Center (CUBIC) zu einer einzigen, nicht redundanten Datenbank verschmolzen worden. Diese Datenbank steht den Entwicklern am CUBIC für ihre Forschung zur Verfügung. Für *Cupe* wurde ein Programm entwickelt, das aus dieser Datenbank eine eigene, binär serialisierte und vollständig indizierte Reaktionsdatenbank extrahiert. Diese Reaktionsdatenbank wird beim Programmstart von *Cupe* dynamisch und sehr schnell geladen. Somit kann der Anwender auf alle benötigten Stoffwechselinformationen – auch ohne Internetverbindung – stets zurückgreifen. Da

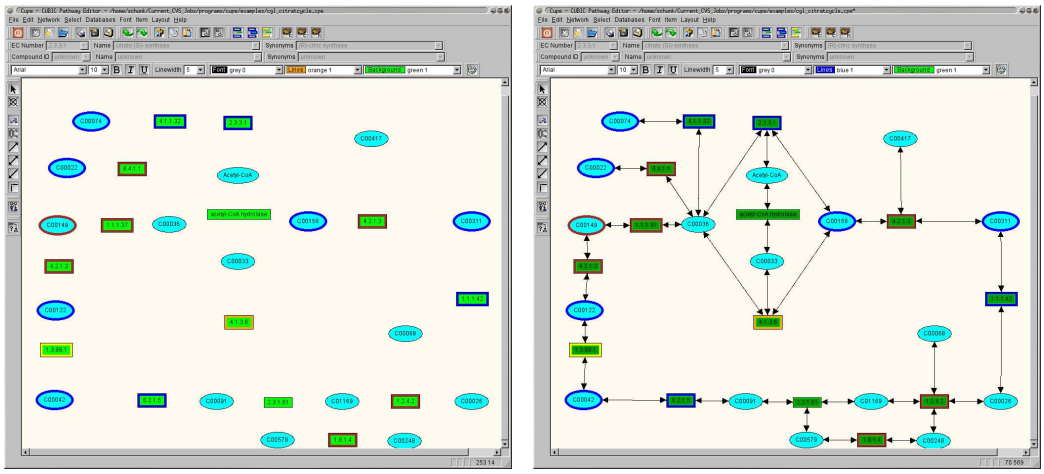


Abbildung 3.39: Analyse der Vernetzung von Metaboliten und/oder Enzymen Vereinigung der CUBIC-Matrix mit der leeren Editor-Matrix mittels *OR*-Operation. In der Abbildung sind zunächst lediglich die für *Corynebacterium glutamicum* annotierten Moleküle des Citratzyklus eingetragen. Durch „einfachen Knopfdruck“ kann die Vernetzung der Moleküle untereinander bestimmt und eingetragen werden.

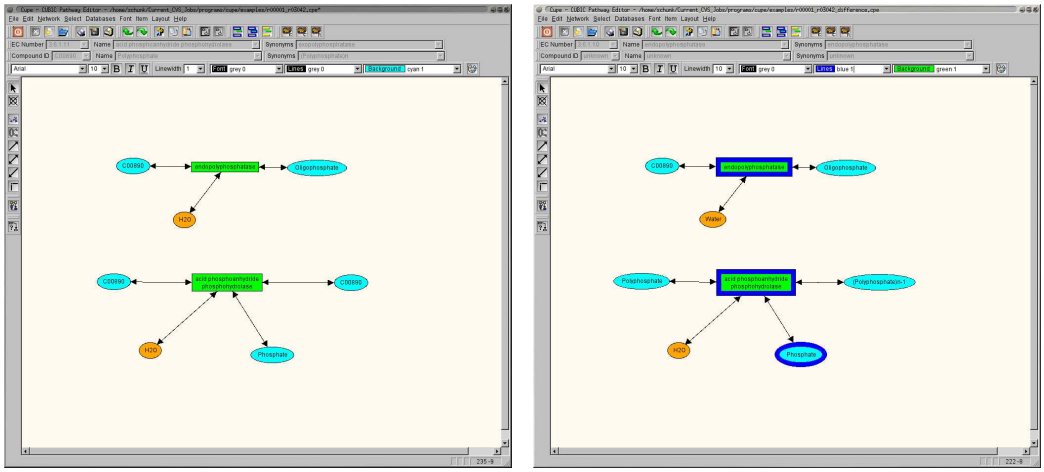


Abbildung 3.40: Unterschiede zwischen zwei Stoffwechselwegen links: Zwei Graphen unterschiedlicher Reaktionen; **rechts:** Die Unterschiede zwischen den beiden Graphen sind blau hervorgehoben. Dieses Beispiel zeigt die Anwendung der *XOR*-Operation.

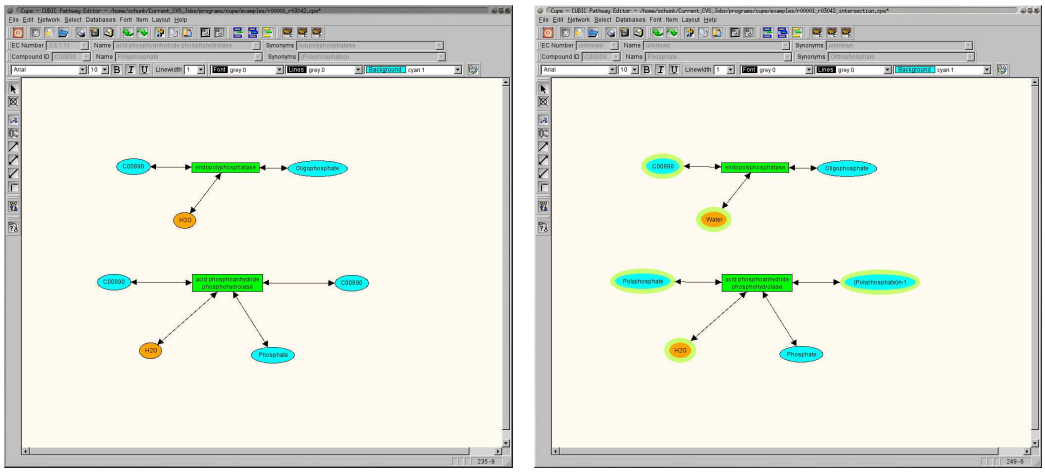


Abbildung 3.41: Gemeinsamkeiten zwischen zwei Stoffwechselwegen links: Zwei Graphen unterschiedlicher Reaktionen; **rechts:** Die Gemeinsamkeiten zwischen den beiden Graphen sind orange hervorgehoben. Dieses Beispiel zeigt die Anwendung der *AND*-Operation.

die *Cupe*-Reaktionsdatenbank bei jedem Programmstart neu eingelesen wird, kann man sie einfach durch eine aktuellere Version austauschen. Aktuelle Versionen der *Cupe*-Reaktionsdatenbank werden über die *Cupe*-Internetseite (siehe 3.8.1 auf Seite 88) angeboten. Dadurch hat man auch den Vorteil, dass stets der Kontakt zu den Anwendern von *Cupe* aufrecht erhalten werden kann.

3.6.2 Sukzessive Netzwerkkonstruktion

Neben den bereits vorgefertigten Sammlungen von Reaktionen für gegenwärtig 310 Biosyntheseprozessen kann ein beliebiges Netzwerk Stück um Stück aufgebaut werden. Hierbei wird der Anwender über die grafische Oberfläche der Datenbankschnittstelle unterstützt. Diese Oberfläche erlaubt es, eine Menge von Reaktionen anhand von Metaboliten oder Enzymen auszuwählen. Anschließend kann diese Menge um weitere Reaktionen erweitert oder ihre Gemeinsamkeiten beziehungsweise Unterschiede zu anderen Reaktionsmengen untersucht werden.

Die Abbildung 3.42 auf der nächsten Seite zeigt, wie beispielsweise erst die Menge der Reaktionen, die von der *Alkoholdehydrogenase* katalysiert werden, ausgewählt wird und diese Menge anschließend auf die Reaktionen **reduziert** wird, die zusätzlich nur *sekundäre Alkohole* umsetzt. In den nächsten Schritten (Abbildung 3.43) wird diese Menge mit der Menge der Reaktionen, die *Ketone* umsetzten, **vereinigt** und abschließend die Reaktionen **entfernt**, die *Zyanide* umsetzen. Die so zusammengesetzte Menge von Reaktionen kann wird abschließend im Editor gezeichnet (siehe Abbildung 3.44 auf Seite 77).

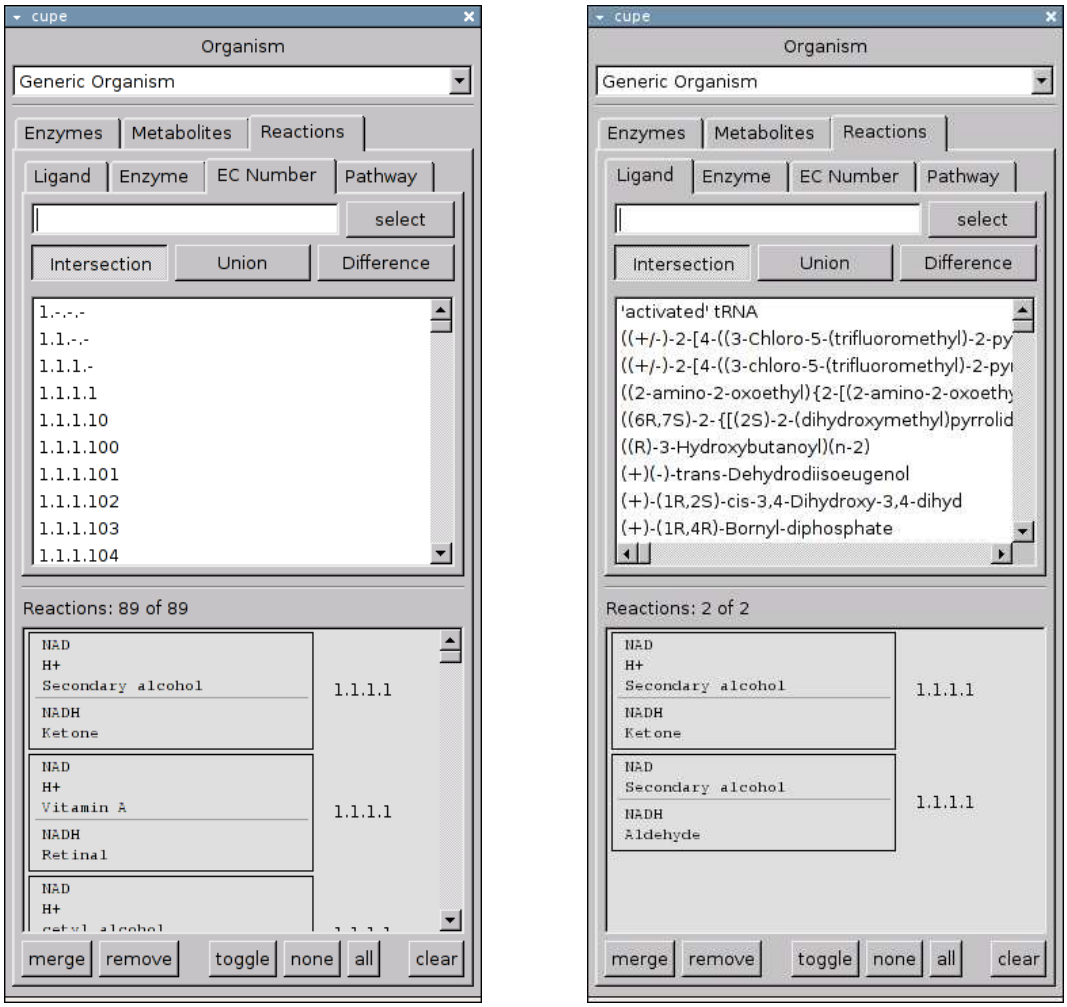


Abbildung 3.42: Sukzessive Auswahl von Reaktionen 1.1.1.1 und Ethanol. **Links**: Im ersten Schritt werden alle Reaktionen ausgewählt, die von der *Alkoholdehydrogenase* katalysiert werden. **Rechts**: Im zweiten Schritt werden über die *Intersection*-Operation nur noch die Reaktionen angezeigt, die zusätzlich auch sekundäre Alkohole umsetzen.

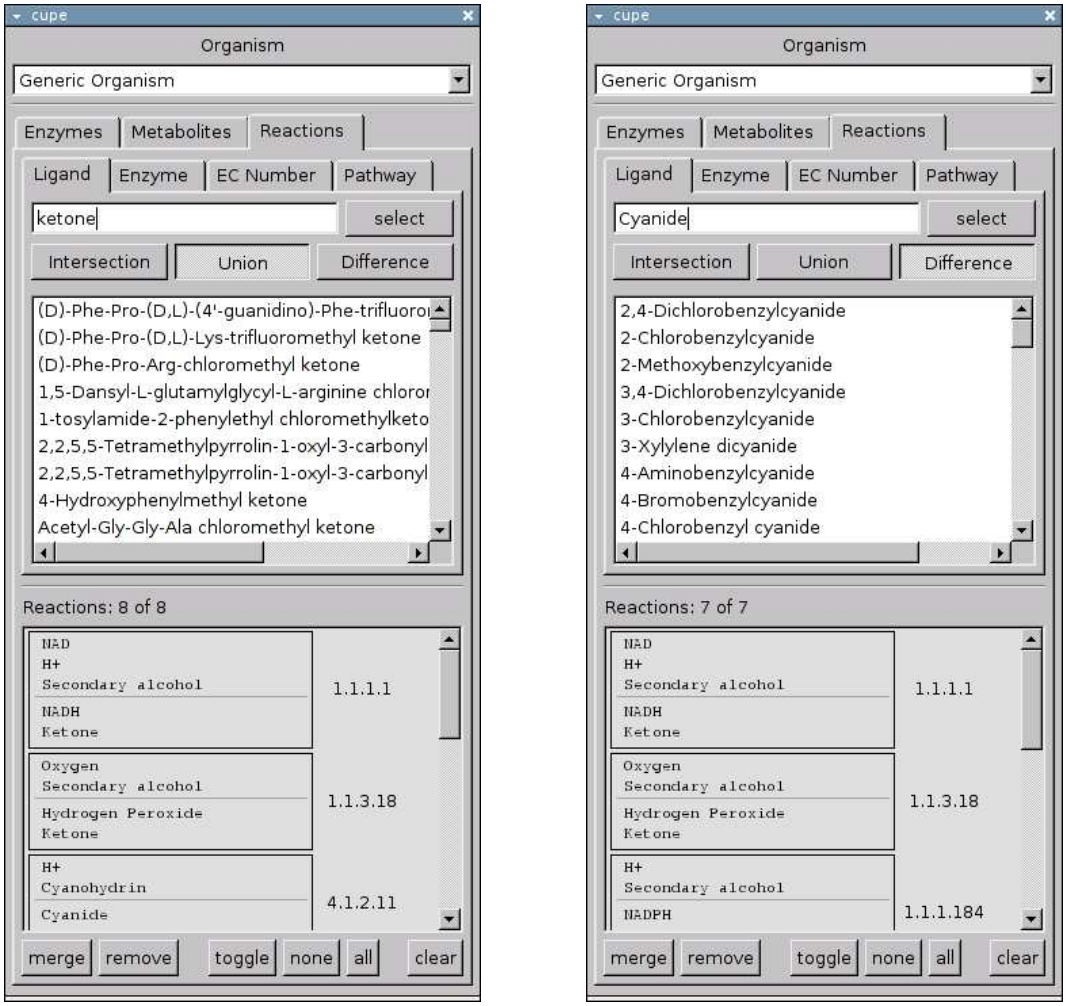


Abbildung 3.43: Ketone aber nicht Zyanide Fortsetzung der sukzessiven Auswahl von Reaktionen (siehe Abbildung 3.42 auf der vorherigen Seite). **Links**: Im dritten Schritt werden über die *Union*-Operation zusätzlich alle Reaktionen hinzugenommen, an denen Ketone beteiligt sind. Schlußendlich werden im vierten und letzten Schritt (**rechts**), mittels der *Difference*-Operation, alle Reaktionen entfernt, die Zyanide umsetzen.

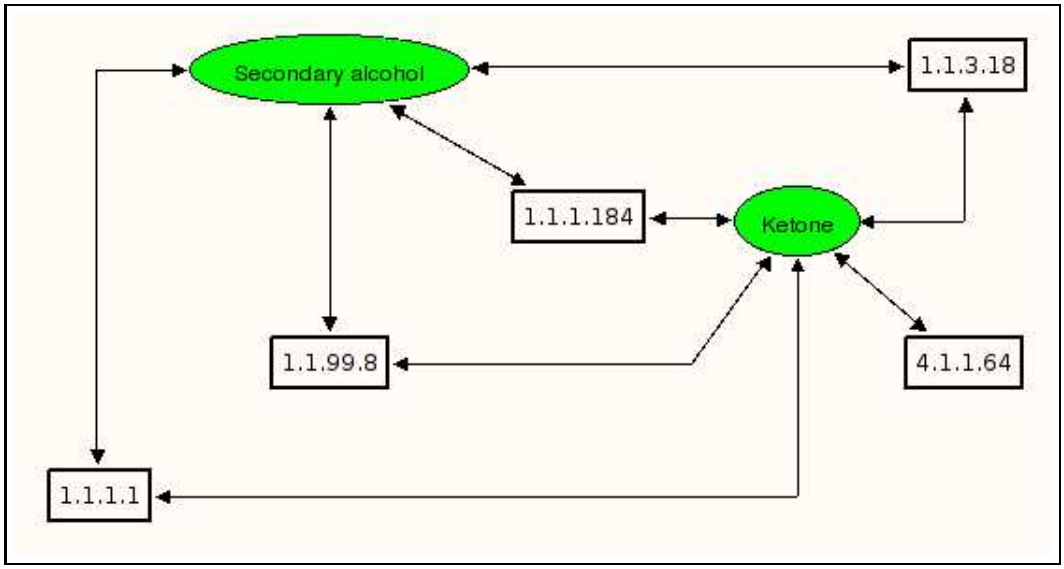


Abbildung 3.44: Sukzessiv aufgebautes Reaktionsnetzwerk Die Reaktionen dieses Graphen wurden sukzessiv durch die Verknüpfung unterschiedlicher Mengenoperationen (vergleiche Abbildungen 3.42 und 3.43) ausgewählt und gezeichnet. Zusätzlich wurden endständige Metabolite in Enzymcluster zusammengefasst (siehe auch Abbildung 3.25 auf Seite 60).

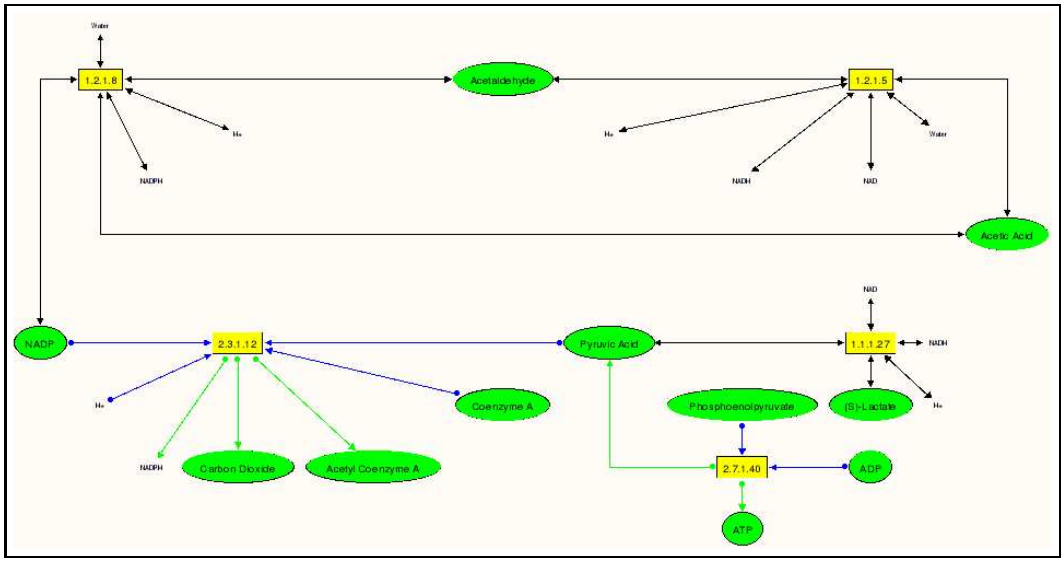


Abbildung 3.45: Kombination von Stoffwechselkarten Schnittstellen-Reaktionen zwischen Glykolyse/Gluconeogenese und Pyruvatstoffwechsel.

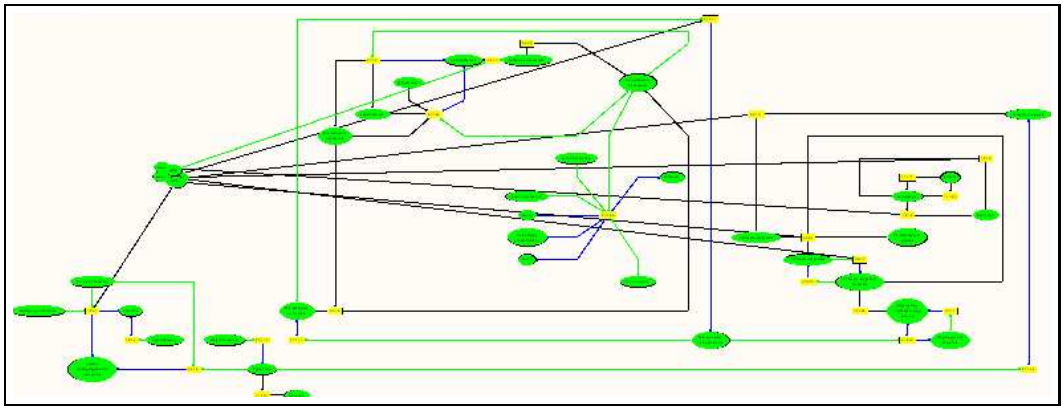


Abbildung 3.46: Vernetzung einzelner Metabolite Cupe ermöglicht es jederzeit, neue Metabolite in einen existierenden Graphen einzufügen und automatisch mit den bereits gezeichneten Reaktionen zu vernetzen. Dieses Beispiel zeigt die Vernetzung eines gepoolten Metaboliten (hier: Wasser). Im Anschluss an die Integration erfolgt in der Regel die Berechnung eines neuen Layouts für den erweiterten Graphen.

3.6.3 Vergleich von Stoffwechselwegen

Die klar abgegrenzten Stoffwechselkarten einzelner Biosyntheseprozesse (Pathways) stellen eine artifizielle Sicht auf den Stoffwechsel dar. Dem unerfahrenen Anwender wird bei solchen statischen Betrachtungen einzelner Pathways (z.B. KEGG-Pathways, Kanehisa & Goto, 1999) suggeriert, dass diese Stoffkreisläufe unabhängig von anderen Reaktionen sind. Zudem lassen sich die Subnetzwerke, die verschiedene Pathways miteinander verbinden, auf diese Weise nicht darstellen. *Cupe* ermöglicht es, solche Verknüpfungen zu erzeugen, darzustellen und mit mehr 4.000 Organismen zu vergleichen (siehe Abbildung 3.45 auf der vorherigen Seite). Diese Technik wird über die gleiche Methodik ermöglicht, wie sie auch für die sukzessive Netzwerkkonstruktion implementiert worden ist.

3.6.4 Vernetzung von Subgraphen

Auf der Grundlage der Mengenoperationen auf metabolischen Netzwerken (3.5) können einzelne Metabolite, Enzyme und beliebige Reaktionen in einen bereits gezeichneten Graphen eingefügt werden (siehe Abbildung 3.47 auf der nächsten Seite). Diese Methode erlaubt es beispielsweise, einen Metaboliten in den Graphen eines Stoffwechselnetzwerkes zu zeichnen und sich auf Knopfdruck dessen Verknüpfungen zum Reaktionsgraphen anzeigen zu lassen (siehe 3.46).

3.7 Programmierschnittstellen

Die Möglichkeit, *Cupe* unabhängig von einem Programmierer oder einer Forschergruppe zu halten, ist ein wesentliches Ziel der Entwicklung. Gerade im akademischen

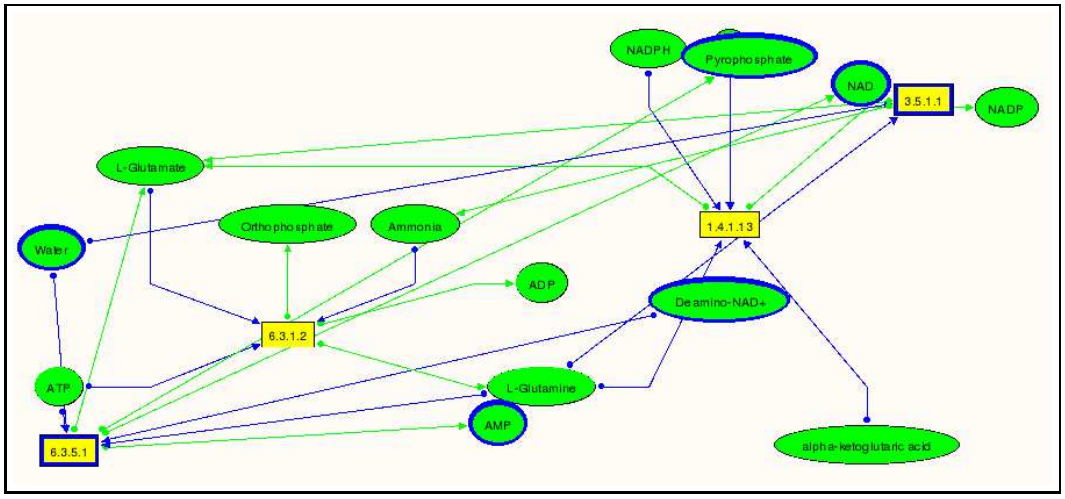


Abbildung 3.47: Subgraphen einfügen Ähnlich wie einzelne Moleküle können auch mehrere Reaktionen (Subgraphen) in ein existierendes Reaktionsnetzwerk eingefügt werden. In dieser Abbildung werden Reaktionen der Enzyme EC 3.5.1.1 und EC 6.3.5.1 zusätzlich verknüpft. Im Anschluss an die Integration erfolgt in der Regel die Berechnung eines neuen Layouts für den erweiterten Graphen.

Umfeld werden durch Praktika, Diplom- und Doktorarbeiten stets neue Personen mit der Fortführung der Entwicklung von *Cupe* betraut. Für dieses Umfeld sind eine Reihe von Programmierschnittstellen entwickelt worden, die auch bereits erfolgreich benutzt werden.

3.7.1 Wichtige Entwurfsmuster

Wie bereits in Abschnitt 2.4.6 erwähnt, werden in *Cupe* eine Vielzahl von wohldefinierten objektorientierten Designmustern benutzt. Wenngleich ihre Implementation der etablierten Technik entspricht, so wurden doch spezielle Anpassungen für *Cupe* vorgenommen.

Befehlsverwaltung

Jede Interaktion mit dem Editor wird über sogenannte Befehle ausgeführt. Jeder Befehl stellt die Funktionen *Rückgängig* und *Wiederherstellen* bereit. Cupe verfügt zur Zeit über einen Befehlssatz von über 180 unterschiedlichen Befehlen. Weitere Befehle können leicht durch die Ableitung von der Klasse *Cupe_Action* implementiert werden. Eine Besonderheit stellt das Befehlsmanagement dar. Im Gegensatz zum „Lehrbuchbeispiel“ ist die Verwaltung der Befehle in der Lage, willkürlich zusammengesetzte Gruppen von Befehlen in sogenannten *Makros* zu kontrollieren (siehe Quellcode 3.1 auf der nächsten Seite). Eine weitere Verbesserung gegenüber der einfachen Befehlsverwaltung stellt die Kontextsensitivität dar. Damit ist gemeint, dass

Quellcode 3.1: Actionmanager Cupe verfügt über eine leistungsstarke und kontextsensitive Befehlsverwaltung. Einzelne Kommandos (Zeile 12) können in Makros zusammengefasst werden (Zeilen 4 und 15). Solchermaßen zusammengefasste Kommandos können jeweils als Einheit verwaltet (u.a. rückgängig gemacht) werden.

```
1  void
2  Cupe_Editor::context_action_delete()
3  {
4      action_manager().start_macro_action();
5
6      typedef Item_Repository::iterator I;
7      I cur = selected_items_.begin();
8      I end = selected_items_.end();
9
10     while ( cur != end )
11     {
12       action_manager().exec( new delete_item( *cur++ ) );
13     }
14
15     action_manager().stop_macro_action();
16 }
```

Befehle auch von anderen Befehlen ausgelöst werden können und dass die so gebildeten Befehlsgruppen automatisch zu Makros gruppiert werden. Diese Technologie ist durch die Entwicklung spezieller Befehle ermöglicht worden, die ihrerseits den Zustand der Befehlsverwaltung auswerten und verändern können.

Zeichenflächenbeobachter

Wie bereits vorgestellt, kann eine Zeichenfläche rekursiv in weitere Zeichenflächen unterteilt sein (siehe 2.4.5, 3.1.1 und 3.2.1). Jede dieser Zeichenflächen kann über eine einheitliche Schnittstelle (*Cupe_Drawing_Area*) angesprochen werden. Gleichzeitig wird über zahlreiche Beobachter (*Observer*) sichergestellt, dass alle von einer Zeichenfläche gelieferten Informationen stets synchron zum aktuellen Zustand der Zeichenfläche sind (siehe Quellcode 3.2 auf der nächsten Seite). Diese Technik ist auch auf die Schnittstelle zu den Algorithmen der Layoutbibliothek AGD ausgedehnt worden, so dass jederzeit das aktuell sichtbare Netzwerk automatisch ausgerichtet werden kann.

Globale Einzelstücke

In *Cupe* gibt es eine Reihe „Globale“ Objekte, die von beliebiger Stelle aus erreichbar sein müssen. Um den Namensraum nicht zu überfrachten und um die fehleranfällige Komplexität durch Seiteneffekt zu vermeiden, werden solche Objekte als Singleton implementiert. Diese Technik garantiert, dass stets nur ein einziges, rechtzeitig und

Quellcode 3.2: Observer Alle Netzwerkelemente und die Verknüpfung unter ihnen werden von zahlreichen *Observern* beobachtet. Zustandsänderungen (Zeilen 4 + 5) werden so automatisch ausgewertet (Zeile 7) und die abhängigen Datenstrukturen synchronisiert. So wird sichergestellt, dass die nachfolgenden Algorithmen mit korrekten Daten arbeiten (Zeilen 13–16).

```
1  void
2  Cupe_Molecule::expand()
3  {
4      std::for_each(
5        composite.begin(), composite.end(), show_composite() );
6
7      inform_observers( new_visibility );
8  }
9
10 const Cupe_Drawing_Area::Molecules &
11 Rectangle_Drawing_Area::molecules() const
12 {
13     if ( expanded_ )
14         return observer().molecules().visible();
15     else
16         return observer().molecules().collapsed();
17 }
```

gültig initialisiertes Objekt einer Singletonklasse existiert. Gerade im Zusammenhang mit dynamisch zur Laufzeit ladbaren Plugins gibt es aber auch Probleme, die „Einzigartigkeit“ eines Singletonobjekts zu garantieren. Die bisher bekannten Komplikationen mit Singletons sind in *Cupe* alle gelöst worden, und die einzelnen Objekte können effektiv eingesetzt werden (siehe Quellcode 3.3 auf der nächsten Seite). Dennoch sollte die Neuentwicklung weiterer Singletons für *Cupe* oder *Cupe*-Plugins besser vermieden werden.

3.7.2 Plugins

Cupe verfügt zwar auch über eigene Analysemethoden, es ist aber für *Cupe* nicht möglich, alle gegenwärtig und zukünftig gewünschten Analysemethoden und Darstellungsformen bereitzustellen. Stattdessen besitzt *Cupe* eine robuste und gut getestete Programmierschnittstelle für die Entwicklung von Plugins. Auf diese Weise ist es auch für andere Entwickler möglich, Erweiterungen für *Cupe* zu schreiben. Plugins, die von anderen Forschern entwickelt werden, profitieren in besonderem Maße davon, dass deren Entwickler Experten für die neue Funktionalität sind. Das heißt, dass auch Methoden zur Analyse von Stoffwechselnetzwerken entwickelt werden können, über die im eigenen Hause kein ausreichendes Wissen vorliegt. Die Koordination solcher Entwicklungen erfolgt über eine offene Internetseite.

Quellcode 3.3: Singleton Das Designmuster `Singleton` garantiert den globalen Zugriff auf die einzige und immer gültige Instanz einer Datenstruktur in jedem beliebigen Programmteil oder Plugin (Zeilen 4–7), ohne dabei die Kontrolle über die Initialisierung der Datenstrukturen zu verlieren (vergleiche Zeilen 13–16).

```
1  void
2  singletons()
3  {
4    parameters = Parameters::get();
5    dictionary = Property_Types::Dictionary();
6    factories  = Cupe_Item_Factories::get_factory_map();
7    ligand_db  = Cupe_Ligand::get_db();
8  }
9
10 Cupe_Ligand*
11 Cupe_Ligand::get_db()
12 {
13     if ( !singleton_ )
14         singleton_ = new Cupe_Ligand();
15
16     return singleton_;
17 }
```

Header

Für die Entwicklung eigener Erweiterungen bietet der Sourcecode von *Cupe* extra ein spezielles Kommando, mit dem man ein vollständiges Paket für die Plugin-Entwicklung erzeugen kann. Der Plugin-Entwickler kann dieses Paket lokal installieren und sich anhand zweier gut dokumentierter Beispiele mit der Entwicklung eigener Plugins vertraut machen. Der wesentliche Schritt ist dabei die Vererbung der sehr einfachen Pluginschnittstelle an die eigene Klasse und die Aufrufung eines vordefinierten Präprozessormakros (siehe Programmbeispiel 3.4 auf der nächsten Seite).

Plugins von Cupe

Über die Pluginschnittstelle sind für Cupe bereits einige Erweiterungen geschrieben worden. Es existiert ein Beispiel-Plugin, das den Zugriff auf die Knoten des Graphen demonstriert, ein Plugin, das statistische Informationen zum Editor ausgeben kann und ein Plugin, um Enzymcluster zu erzeugen (siehe Abbildung 3.48 auf Seite 84).
Ein besonders umfangreiches Plugin ist das *Graph Comparison Plugin*. Dieses Plugin ermöglicht es, zwei Graphen mittels der neu definierten Mengenoperationen (siehe Abschnitt 3.5) zu vergleichen und die Ergebnisse entsprechend der Fragestellung farbig zu kennzeichnen (siehe Abbildungen 3.40 + 3.41). Dieses Plugin verfügt über eine

Quellcode 3.4: Cupe Info Plugin Die Programmierschnittstelle für neue Plugins von Cupe kann einfach durch Vererbung implementiert werden (Zeile 1). Das Beispiel zeigt die vollständige Deklaration des *Cupe Info Plugins* (siehe Abbildung 3.48 auf der nächsten Seite rechts). Um einen sicheren Zugriff von Cupe auf das Plugin zu gewährleisten, ist lediglich eine zusätzliche Anweisung in Form eines Makros notwendig (Zeile 26).

```
1  class Cupe_Info_Plugin : public Plugin
2  {
3      Q_OBJECT
4      private:
5          Cupe_Statistics*     window_;
6          QAction*             starter_;
7
8      public:
9          Cupe_Info_Plugin();
10         ~Cupe_Info_Plugin();
11
12         QString plugin_name() const;
13         QString description() const;
14         QString developer()   const;
15         QString date()        const;
16         QString bug_reports() const;
17
18         QAction* menu_entry()     const;
19         QAction* toolbar_entry() const;
20
21     public slots:
22         void salute();
23         void update();
24 };
25
26 CUPE_PLUGIN( Cupe_Info_Plugin )
```

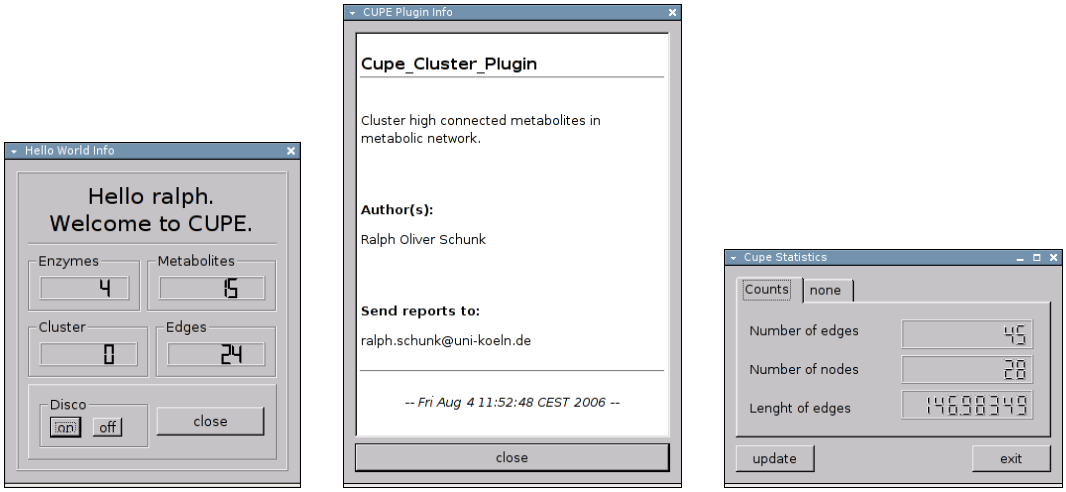


Abbildung 3.48: Einfache Plugins Über die Pluginschnittstelle von Cupe ist es sehr einfach, eigene Plugins zu programmieren und in Cupe zu benutzen. **Links**: Das „Discoplugin“ dient dazu, den Zugriff auf die Elemente eines Editors zu demonstrieren. **Mitte**: Zu jedem Plugin wird automatisch ein kleiner Informationsdialog bereitgestellt. **Rechts**: Ein kleines Plugin, das statistische Informationen liefert.

umfangreiche Dokumentation und eine komfortable grafische Benutzerschnittstelle (siehe Abbildung 3.49 auf der nächsten Seite).

Externe Plugins

Die Pluginschnittstelle von *Cupe* wird auch schon von anderen Wissenschaftlern des Instituts genutzt (siehe Abbildung 3.50 auf Seite 87). Für die Darstellung der unterschiedlichen Metabolitkonzentrationen wurde das *Rationindikator*-Plugin entwickelt (Thielen, B., 2005). Die Darstellung und Simulation von Stoffwechselsystemen mittels Petri-Netzwerken wird über das *MPN*-Plugin ermöglicht (Hartman, K., 2005).

3.7.3 Zeichenflächen

Jedes Netzwerkelement, das innerhalb des Editors gezeichnet wird, ist einer Zeichenfläche zugeordnet (siehe Quellcode 3.5 auf Seite 86). In *Cupe* sind zwei Arten von Zeichenflächen implementiert worden, die des Boards (3.1.1) und die der Cluster (3.2.1). Darüber hinaus sind aber auch noch weitere Spezialisierungen der Klasse *Cupe_Drawing_Area* denkbar, etwa um runde Zeichenflächen zu konstruieren. Zeichenflächen haben die wesentlichen Vorteile, dass sie einen kontrollierten Zugriff auf ihre Elemente bieten (Quellcode 3.2), dass man ihr Layout separat berechnen und dass man ihre Größe und Position verändern kann, ohne dabei den Subgraphen, den sie beherbergen, zu verändern (siehe Abbildungen 3.26 bis 3.29).

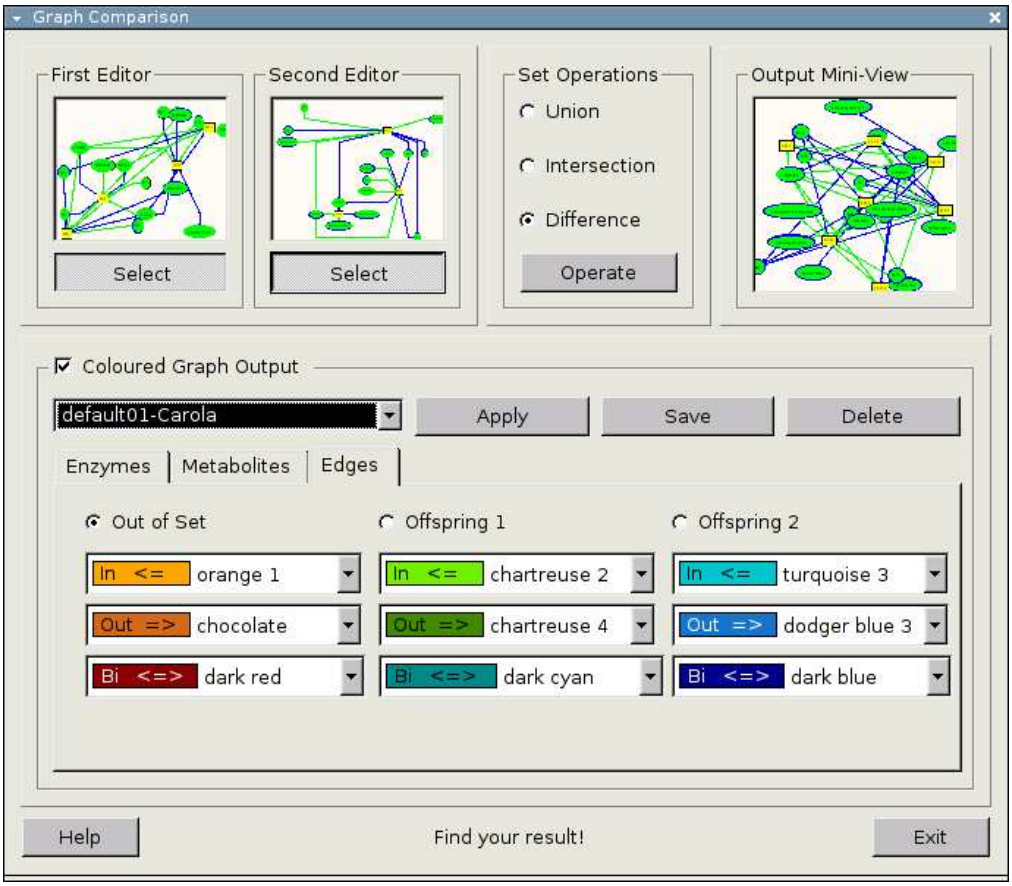


Abbildung 3.49: Graph Comparison Plugin Das *Graph Comparison Plugin* erweitert Cupe um eine Benutzerschnittstelle zum Vergleichen metabolischer Netzwerke. Dieses Plugin zeigt, dass auch sehr komplexe Zugriffe und Analysen über die Pluginschnittstelle möglich sind.

Quellcode 3.5: Zeichenflächen Zeichenflächen gehören zu den wichtigsten Abstraktionen in der Cupe-Programmierung. Die virtuelle Deklaration (Zeilen 1–4) wird durch Ableitung in *Cupe_Board* und *Rectangle_Drawing_Area* überschrieben und definiert (Zeilen 18 und 23). Jedes Netzwerkelement (`Cupe_Item`) befindet sich immer unter der Kontrolle einer Zeichenfläche (Zeile 12). Es ist aber möglich, in andere Zeichenflächen zu wechseln, auch wenn diese nicht sichtbar sind.

```
1  class Cupe_Drawing_Area
2  {
3    /* Deklaration der virtuellen Zeichenfläche */
4  };
5
6  typedef boost::shared_ptr<Cupe_Drawing_Area> Drawing_Area_ptr;
7
8  class Cupe_Item: public QObject, boost::noncopyable
9  {
10     ...
11      public:
12         Cupe_Item( Drawing_Area_ptr );
13     ...
14 };
15
16 class Cupe_Board : public QCanvas, public Cupe_Drawing_Area
17 {
18   /* Deklaration der Zeichenfläche eines Boards */
19 };
20
21 class Rectangle_Drawing_Area : public Cupe_Drawing_Area
22 {
23     /* Deklaration der Zeichenfläche in Molekülclustern */
24 };
```

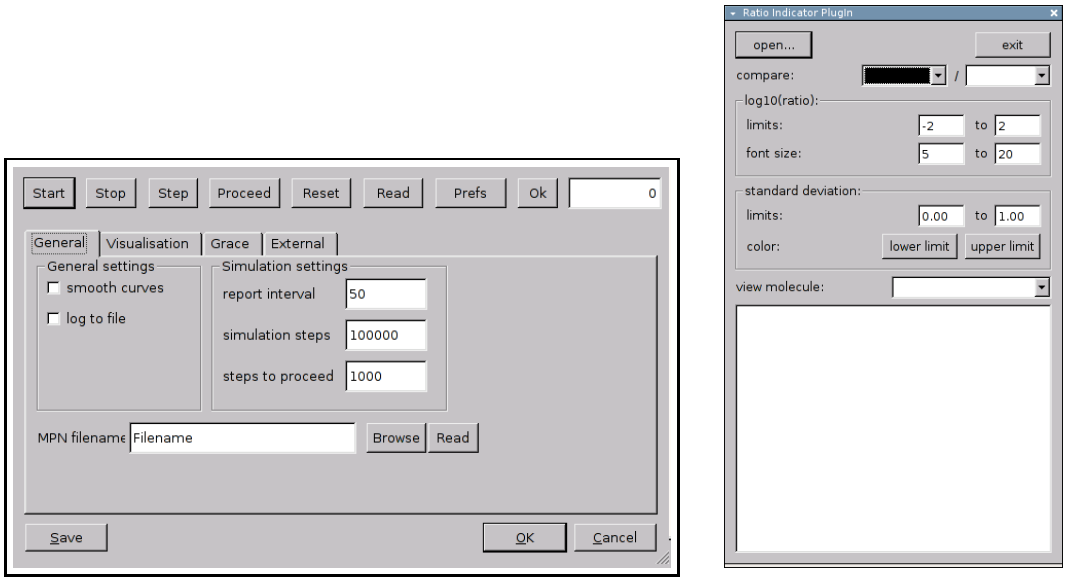


Abbildung 3.50: Externe Plugins Die Pluginschnittstelle von Cupe wird auch von anderen Wissenschaftlern genutzt. **Links**: Das Plugin „Metabolic Petri Net“ dient zur Darstellung der Ergebnisse von Petri-Netz Simulationen auf metabolischen Netzwerken. **Rechts**: Das „CUPE Ratio Indicator Plugin“ kann die Konzentrationsänderungen von Metaboliten zwischen unterschiedlichen Stoffwechselzuständen wiedergeben.

3.7.4 Speichern und Lesen von Stoffwechselnetzwerken

Cupe benutzt die selbst entwickelte Auszeichnungssprache *CupeML* (Version 1.3.16) zum Speichern und Lesen von Dateien (siehe Beispiel 3.6 auf Seite 89). Dieses lesbare XML-Dateiformat wird von der Qt-Bibliothek systemübergreifend unterstützt. Das *CupeML*-Format ist sehr ausführlich, damit der Aufbau einer Datei selbsterklärend bleibt (siehe 2.4.2). Der Nachteil dieses Formates ist jedoch, dass es sehr viel Speicherplatz benötigt. Es ist jedoch möglich, die Dateien komprimiert zu speichern und so den Speicherbedarf zu reduzieren. Da das Dateiformat häufig angepasst wurde und ältere Dateien dennoch lesbar bleiben sollten, wird die Version des benutzten *CupeML*-Formats in jeder *Cupe*-Datei (*.cpe) mit angegeben. *Cupe* kann dann beim Einlesen von gespeicherten Netzwerken die richtigen Importfilter auswählen.

Zusätzliche Daten speichern

Für die Entwicklung eigener Plugins ist es sehr hilfreich, wenn Plugins zusätzliche Informationen zu einem metabolischen Netzwerk speichern können. Für diesen Zweck ist eine sehr einfach zu bedienende Programmierschnittstelle entwickelt worden, die es erlaubt, beliebige Zusatzinformationen zu speichern (siehe Programmbeispiel 3.7 auf Seite 90). Die einzige Bedingung, die diese Extradaten erfüllen müssen, ist, dass sie sich in Zeichenketten umwandeln lassen und auch wieder aus Zeichenketten eingelesen werden können. Das Dateiformat von *Cupe* ist dabei so robust, dass Dateien

Tabelle 3.7: Statistische Werte zum Quellcode von *Cupe*

Quellcode	Anzahl
Dateien	411
Klassen	647
Funktionen	6101
Zeilen	71833

mit zusätzlichen Informationen auch dann gelesen werden können, wenn das Plugin nicht zur Verfügung steht. Je nach Art der zusätzlichen Daten werden diese dann entweder nicht gelesen oder können während des Programmablaufes nicht benutzt werden. Die weitere Funktionalität von *Cupe* bleibt aber vollständig erhalten.

3.7.5 Statistisches

Der Umfang, den die Entwicklung einer grafischen Anwendung wie *Cupe* annehmen kann, wird leicht unterschätzt. Ein wohl durchdachtes Design und konsistente Organisation des Quellcodes sind für eine erfolgreiche Projektdurchführung unerlässlich (vergleiche Tabelle 3.7).

3.8 Internetpräsenz

Neue Methoden und Entwicklungen in der Bioinformatik werden fast immer über Fachzeitschriften und Internetseiten publiziert. Aber auch für die institutsinterne Verteilung werden zunehmend interne Server eingesetzt. Die Etablierung neuer Methoden hängt daher wesentlich von ihrer Verfügbarkeit und Kommunikation über das Internet ab.

3.8.1 Cupe Knowledge Portal

Die Anforderungen an eine Internetpräsenz von *Cupe* entstammen vor allem aus dem universitären Umfeld und der Interdisziplinärität der Entwickler und Anwender von *Cupe*. Damit die Entwicklung auch über Jahre hinweg erfolgreich verläuft, ist es nötig, alle Techniken für die Diskussion und Entwicklung rund um *Cupe* und dessen Plugins einheitlich und integriert in einer Webseite bereitzustellen. Dabei ist es besonders wichtig, dass sowohl der Aufwand für die zentrale Bereitstellung der Technologien als auch die Barrieren zur Pflege der Inhalte besonders gering und gut dokumentiert sind. Das Kommunikationsmodell für die Weiterentwicklung und Verteilung von *Cupe* (siehe Abbildung 3.51) folgt deswegen dem gleichen Modell wie ein Wikinetzwerk[3]. Jede Entwicklung für *Cupe* kann hier diskutiert und schließlich dokumentiert werden (siehe Abbildung 3.52). Im Rahmen einer von der Universität durchgeführten Veranstaltung (Projekte, Praktika, Diplom-, Doktorarbeiten usw.)

[3] z.B. www.wikipedia.de

Quellcode 3.6: Dateiformat *Cupe* benutzt die selbst entwickelte Auszeichnungssprache *CupeML 1.3.16* Diese Sprache ist aufgrund ihrer Ausführlichkeit nahezu selbsterklärend. Dadurch ist für andere Entwickler leicht zu verstehen, wie die Dateien aufgebaut sind; es wird aber auch deutlich mehr Speicherplatz gebraucht, als in anderen Formaten nötig wäre. Diese Datei speichert z.B. „nur“ einen Graphen aus vier Knoten und drei Kanten.

```
<!DOCTYPE CupeML>
<CupeML subversion="3" subsubversion="16" version="1" >
 <Board width="255" expanded="1" boardname="example.cpe" height="216" pool_level="0" >
  <Clusters count="0" />
  <Enzymes count="1" >
   <Item x="72" y="106" type="4060086272" id="185780064" state="VISIBLE" father="0" >
    <Property type="QColor" value="#ffff00" name="Background color" />
    <Property type="QString" value="3.2.1.28" name="Displayed name" />
    <Property type="QFont" value="(Helvetica) 9 50 0 0" name="Font" />
    <Property type="QColor" value="#000000" name="Font color" />
    <Property type="QColor" value="#0000ff" name="Foreground color" />
    <Property type="int" value="4" name="Line width" />
    <Property type="Molecule_Size" value="1 0 56 23" name="Size" />
   </Item>
  </Enzymes>
  <Metabolites count="3" >
   <Item x="72" y="44" type="4043309056" id="181176632" state="VISIBLE" father="0" >
    <Property type="QColor" value="#00ff00" name="Background color" />
    <Property type="QString" value="Trehalose" name="Displayed name" />
    <Property type="QFont" value="(Helvetica) 9 50 0 0" name="Font" />
    <Property type="QColor" value="#000000" name="Font color" />
    <Property type="QColor" value="#0000ff" name="Foreground color" />
    <Property type="int" value="4" name="Line width" />
    <Property type="Molecule_Size" value="1 0 87 36" name="Size" />
   </Item>
   <Item x="182" y="106" type="4043309056" id="181248688" state="VISIBLE" father="0" >
    <Property type="QColor" value="#00ff00" name="Background color" />
    <Property type="QString" value="D-Glucose" name="Displayed name" />
    <Property type="QFont" value="(Helvetica) 9 50 0 0" name="Font" />
    <Property type="QColor" value="#000000" name="Font color" />
    <Property type="QColor" value="#0000ff" name="Foreground color" />
    <Property type="int" value="4" name="Line width" />
    <Property type="Molecule_Size" value="1 0 91 36" name="Size" />
   </Item>
   <Item x="72" y="168" type="4043309056" id="181251808" state="VISIBLE" father="0" >
    <Property type="QColor" value="#00ff00" name="Background color" />
    <Property type="QString" value="Water" name="Displayed name" />
    <Property type="QFont" value="(Helvetica) 9 50 0 0" name="Font" />
    <Property type="QColor" value="#000000" name="Font color" />
    <Property type="QColor" value="#0000ff" name="Foreground color" />
    <Property type="int" value="4" name="Line width" />
    <Property type="Molecule_Size" value="1 0 57 36" name="Size" />
   </Item>
  </Metabolites>
  <Edges count="3" >
   <Item Points="" from="185780064" type="15859712" to="181176632" >
    <Property type="QColor" value="#ffffff" name="Background color" />
    <Property type="QString" value="To source edge" name="Displayed name" />
    <Property type="QFont" value="(Helvetica) 9 50 0 0" name="Font" />
    <Property type="QColor" value="#000000" name="Font color" />
    <Property type="QColor" value="#0000ff" name="Foreground color" />
    <Property type="int" value="1" name="Line width" />
    <Property type="Molecule_Size" value="1 0 0 0" name="Size" />
   </Item>
   <Item Points="" from="185780064" type="15794176" to="181248688" >
    <Property type="QColor" value="#ffffff" name="Background color" />
    <Property type="QString" value="To target edge" name="Displayed name" />
    <Property type="QFont" value="(Helvetica) 12 50 0 0" name="Font" />
    <Property type="QColor" value="#000000" name="Font color" />
    <Property type="QColor" value="#00ff00" name="Foreground color" />
    <Property type="int" value="1" name="Line width" />
    <Property type="Molecule_Size" value="1 0 0 0" name="Size" />
   </Item>
   <Item Points="" from="185780064" type="15859712" to="181251808" >
    <Property type="QColor" value="#ffffff" name="Background color" />
    <Property type="QString" value="To source edge" name="Displayed name" />
    <Property type="QFont" value="(Helvetica) 9 50 0 0" name="Font" />
    <Property type="QColor" value="#000000" name="Font color" />
    <Property type="QColor" value="#0000ff" name="Foreground color" />
    <Property type="int" value="1" name="Line width" />
    <Property type="Molecule_Size" value="1 0 0 0" name="Size" />
   </Item>
  </Edges>
 </Board>
</CupeML>
```

Quellcode 3.7: Properties Werte eigener Datentypen können einfach in die *Properties* eines Netzwerkelements geschrieben werden (Zeilen 6–9). Um diese Werte automatisch zu speichern bzw. wieder einzulesen, muss ihr Datentyp lediglich registriert werden (Zeilen 3 + 4).

```
1  void properties_example()
2  {
3    Property_Types* dictionary = Property_Types::Dictionary();
4    dictionary->register_type<QColor>( "QColor" );
5
6    Cupe_Item* item        = new Cupe_Enzyme( drawing_area_ );
7    Properties p           = item->properties();
8    p["Low ratio color"]   = QColor( Qt::green );
9    item->properties( p );
10 }
```

ist es durchaus gerechtfertigt, dass die Inhalte der Veranstaltung über dieses *Cupe Knowledge Portal* dokumentiert werden müssen. Andernfalls sollte die Bescheinigung einer erfolgreichen Teilnahme verweigert werden, da die Pflege der Inhalte über eine zentral verantwortliche Stelle kaum über Jahre hinweg aufrecht erhalten werden kann und deshalb ausdrücklich nicht erwünscht ist.

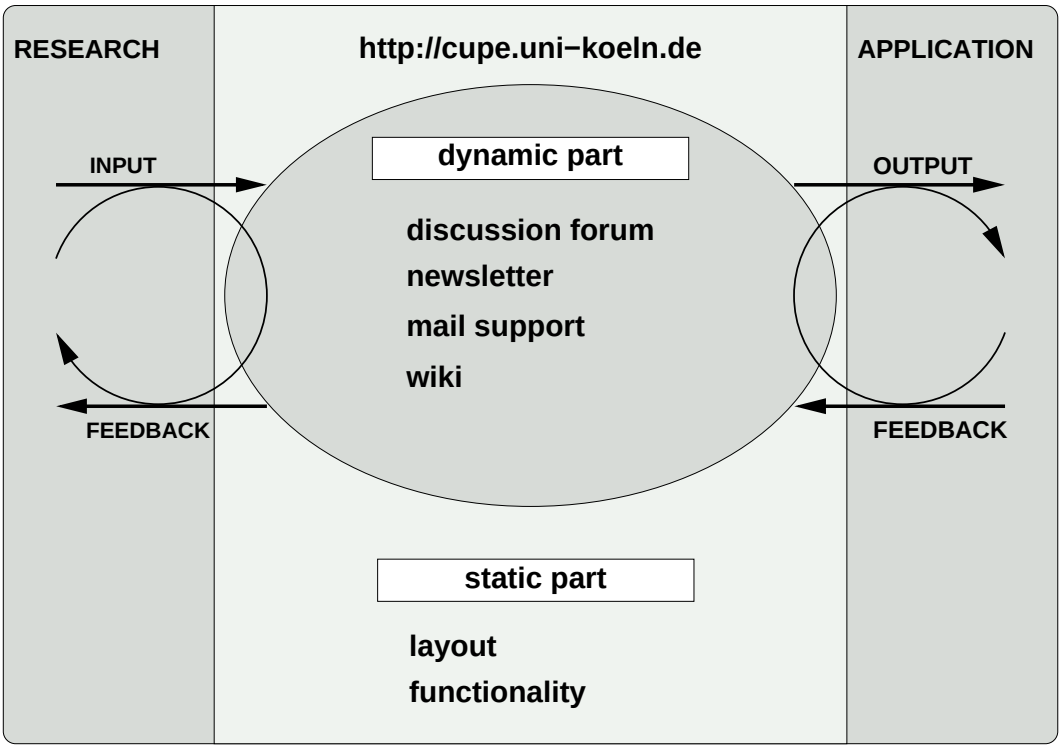


Abbildung 3.51: Projektkommunikation Die gesamte Kommunikation zwischen Entwicklern und Anwendern von *Cupe* gleicht einem rückgekoppelten Kreislauf. Alle Inhalte werden dynamisch von allen Benutzern erzeugt und können direkt bereitgestellt und diskutiert werden. Die zentrale Verwaltung ist auf ein Mindestmaß reduziert worden.

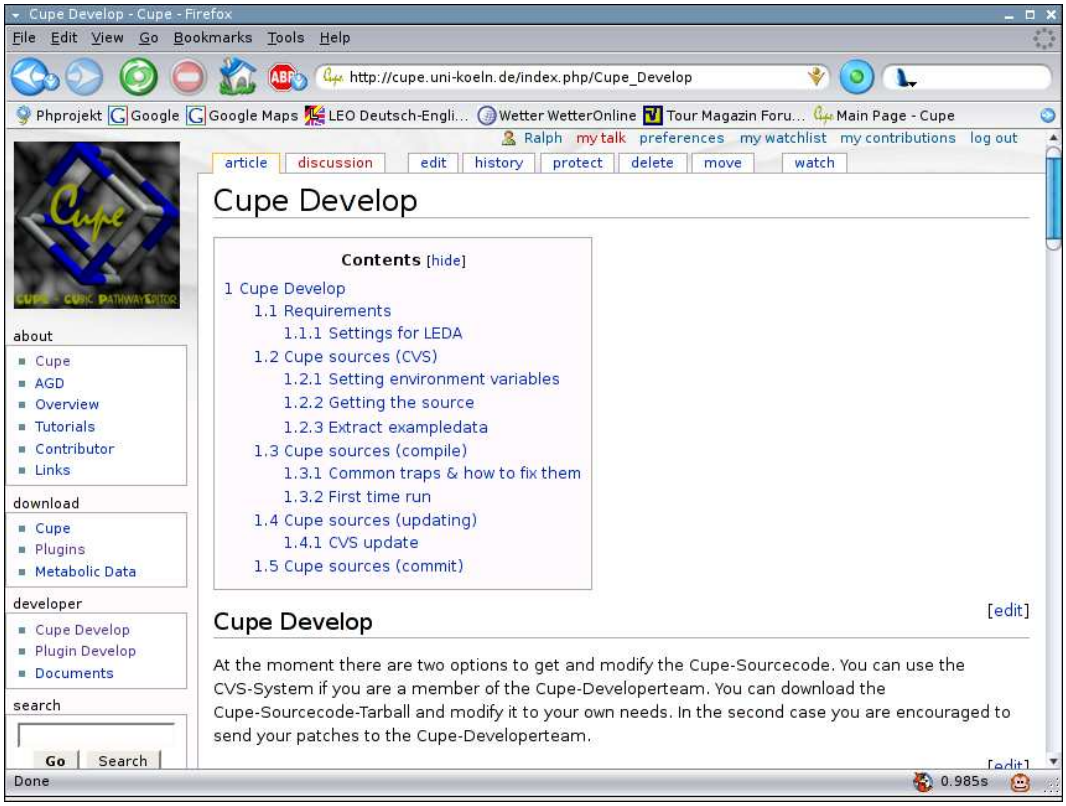


Abbildung 3.52: Cupe Knowledge Portal Die Internetseite zu Cupe dient Entwicklern und Anwendern als Kommunikationsplattform. Das *Cupe Knowledge Portal* baut auf der gleichen Technologie wie Wikipedia auf; dadurch können neue Beiträge oder Erweiterungen jederzeit und nahezu barrierefrei von allen Benutzern eingetragen werden.

4 Diskussion

In der Bioinformatik sind mittlerweile eine Vielzahl von Methoden für die Stoffwechselanalyse entwickelt worden. Mit *Cupe* existiert jetzt ein Programm, das auf einzigartige Art und Weise die Generierung, Darstellung und Analyse von Stoffwechselnetzwerken mit den aktuellsten Methoden zum automatischen Zeichnen von Graphen verbindet.

4.1 Cupe als Editor für metabolische Netzwerke

Cupe unterstützt sowohl die manuelle wie auch die automatische Erzeugung von metabolischen Netzwerken. Besonders die Möglichkeit, alle Netzwerkelemente völlig frei zu formatieren, hebt *Cupe* gegen die breite Masse der bereits verfügbaren Netzwerkeditoren ab (siehe Abbildung 3.15). Lediglich Programme wie „Java editor for biological pathways“ oder „Cytoscape“ können die gleichen Freiheitsgrade bei der Formatierung der Netzwerkelemente aufweisen, dafür kann man mit ihnen Netzwerke aber nicht automatisch konstruieren.
Absolut einzigartig ist die Fähigkeit von *Cupe*, Subnetzwerke in Superknoten bzw. Clustern (Abschn. 3.2.1) zusammenzufassen und einen Metabolitpool (Abschn. 1.3.1) zu verwalten. Beide Methoden zusammen erlauben es, den Graphen eines Reaktionsnetzwerks erheblich zu vereinfachen und so die Lesbarkeit automatisch gezeichneter Graphen deutlich zu verbessern (Tabelle 3.1). Diese Verfahren kommen dabei ohne Informationsverlust oder hohe Abstraktionen aus (vergleiche Abbildung 1.5). Derzeit wird das Pooling von Metaboliten ausschließlich automatisch aus dem Vernetzungsgrad der Metabolite berechnet. Für eine problemorientiertere Darstellung von Netzwerken wäre es aber auch wünschenswert, wenn der Anwender die Metabolite, die im Graphen gepoolt dargestellt werden sollen, selber auswählen kann.
Die Tatsache, dass die Teilnetzwerke in Clustern bei der Neuberechnung des Layouts unverändert bleiben, erleichtert es zudem, sich nach einer Neuberechnung des Layouts schnell wieder zu orientieren und stellen somit erste Ansätze zur Bereitstellung einer *Mentalen Karte* dar (Abschn. 1.1.2).
Die außerordentlich hohe Flexibilität von *Cupe* bei der Darstellung von metabolischen Netzwerken und die zahlreichen Möglichkeiten der interaktiven Manipulation führen aber auch dazu, dass sich ein nicht angeleiteter Anwender die volle Funktionalität von *Cupe* nur langsam erarbeiten kann. Hier könnte das Befehlsmanagement von *Cupe* Abhilfe schaffen (Abschn. 3.7.1). Die Fähigkeit der Befehlsverwaltung, mehrere Befehle in Makros zusammenzufassen, könnte dazu genutzt werden, vorgefertigte Makros anzubieten, um beispielsweise einfache Analysen zu automatisieren.

Auch die Darstellung sehr großer Reaktionsnetzwerke wird von *Cupe* mittels vieler gleichzeitig verfügbarer Ansichten – auch in unterschiedlichen Vergrößerungsstufen – auf ein und den selben Graphen vereinfacht (Abschn. 3.1.1). Andere Programme zur Darstellung von Netzwerken unterstützen höchstens eine – oft frei skalierbare – Ansicht. Wenn nur eine Ansicht bereitsteht, ist jedoch der direkte Vergleich verschiedener Ausschnitte eines Netzwerkes nur sehr umständlich durchzuführen. *Cupe* ist für derartige Betrachtungen deutlich besser geeignet. Die Größe der Graphen, die in *Cupe* gezeichnet werden können, ist allerdings durch den hohen Speicherbedarf für die einzelnen Netzwerkelemente beschränkt. Dieses Speicherverhalten begrenzt die Größe der Netzwerke auf ca. 1.500 Knoten und Kanten. Da die verfügbaren Layoutalgorithmen aber noch nicht in der Lage sind, größere Netzwerke biologisch sinnvoll auszurichten, stellt dieses keine echte Einschränkung der Funktionalität von *Cupe* dar.

4.2 Cupe für die Analyse von metabolischen Netzwerken

Dank eines vorprozessierten Datenbestands besitzt *Cupe* unter allen bisher bekannten Programmen für die Stoffwechselanalyse die größte Reaktionsdatenbank. Die über 32.000 Einträge stehen im Gegensatz zu „ExPASy“ oder „BioPath“ ohne direkte Internetverbindung zur Verfügung und können über eine Internetseite jederzeit aktualisiert werden. Mit Hilfe der leistungsstarken, grafischen Datenbankschnittstelle (Abschn. 3.1.3) können die Anwender von *Cupe* diese Reaktionen zu jedem beliebigen Netzwerk zusammenstellen (Abschn. 3.6.2). Die Darstellung von Pathways ist dabei nicht nur auf die Bereitstellung von vorgefertigten Sammlungen beschränkt (vergleiche „KEGG“), sondern kann ebenfalls frei kombiniert werden (Abschn. 3.6.3).
Mit der Entwicklung der „Mengenlehre für metabolische Netzwerke“ bietet *Cupe* eine einzigartige Schnittstelle (Abschn. 3.5) zum Vergleichen unterschiedlicher Netzwerke. Der Anwender wird dabei durch ein eigens zu diesen Zweck entwickeltes Plugin unterstützt. Dieses „Network Compare Plugin“ bietet dank einer grafischen Benutzerschnittstelle zahlreiche Konfigurationsmöglichkeiten, über die Art der Vergleiche und die Darstellung der Ergebnisse gesteuert werden kann. Leider können Teilnetzwerke, die in Clustern gezeichnet sind, bei diesen Vergleichen noch nicht berücksichtigt werden, da beide Techniken noch nicht kompatibel zueinander sind. Da jede Zeichenfläche aber über *Observer* (Abschn. 3.7.1) verfügt, müsste die Integration der „Mengenlehre für metabolische Netzwerke“ mit der Technologie der Cluster über die Vereinigung der betroffenen *Observer* möglich sein.
Das Spektrum der Analysemethoden von *Cupe* kann mit Hilfe von Plugins für jeden Bedarf erweitert werden. Diese Plugin-Schnittstelle kann ohne detaillierte interne Kenntnisse von *Cupe* leicht von anderen Entwicklern zur Implementation eigener Methoden genutzt werden. Die Tatsache, dass diese Schnittstelle bereits für die Entwicklung neuer Methoden genutzt wurde (Abschn. 3.7.2), zeigt, dass sich *Cupe* in diesem Bereich sehr gut mit Programmen wie „Cytoscape“ und „VANTED“ vergleichen kann.

4.3 Cupe als Ausbildungsplattform

Leider können viele erfolgreiche Projekte der Bioinformatik nach dem Weggang der Hauptentwickler nicht mehr weitergeführt werden. Häufig sind das Expertenwissen und die benötigten technischen Fähigkeiten so stark mit dem Erstentwickler verbunden, dass die Einstiegshürden für neue Mitarbeiter zu hoch werden, um mit vertretbarem Aufwand und in absehbarer Zeit mit dem Projekt vertraut zu werden. Eines der Hauptziele der Entwicklung von *Cupe* war es, den Erstentwickler „überflüssig“ zu machen. Um dieses Ziel zu erreichen, wurde bei der Entwicklung des Quellcodes (Abschn. 2.4.2), bei den benutzten Programmiertechniken (Abschn. 2.4.6) und der Bereitstellung der Entwicklerplattform im Internet stets darauf geachtet, dass gut etablierte und umfangreich dokumentierte Methoden verwendet werden. Auf der Grundlage dieser *akademischen* Softwareentwicklung konnten bereits erfolgreich fünf studentische Projekte, mit direktem Bezug zu *Cupe*, durchgeführt werden. Es ist zwar bei jedem dieser Projekte nötig gewesen, die Studenten anfangs in die entsprechenden Techniken einzuführen, es war ihnen aber im Anschluss daran möglich, neue Konzepte selbständig zu erarbeiten und eigenständige Lösungen zu liefern. Es ist zu erwarten, dass *Cupe* auch in Zukunft als Plattform für weitere interdisziplinäre Projekte, Diplomarbeiten und Dissertationen dienen kann.
Die eigens für die Publikation, Kommunikation und Distribution von *Cupe* eingerichtete Webseite folgt dem Kommunikationsmodell eines Wiki-Netzwerks (Abschn. 3.8.1). Hierdurch ist es für jeden interessierten Anwender und Entwickler sehr leicht möglich, neue Beiträge zu schreiben und an Diskussionen teilzunehmen. Die Erweiterung der *Cupe*-Internetseite ist so ohne eine zentral verantwortliche Person möglich, für die Einrichtung dieser Dienste auf einen Server-Computer ist aber ein Administrator mit umfangreichen Kenntnissen nötig. Im Zuge der Entwicklung von *Cupe* wurde zwar versucht, auch diese Dienste weitgehend zu dezentralisieren, diese Arbeiten konnten aber bisher nicht erfolgreich abgeschlossen werden. Erste Ansätze zeigen aber, dass es prinzipiell möglich ist, diese Dienste dezentral zu „klonen“ und auf anderen Servern zu installieren.

4.4 Weiterentwicklung von Cupe

Cupe ist offen und flexibel programmiert und kann problemlos um weitere Module erweitert werden. Damit *Cupe* für die Zukunft aber auch wirklich gerüstet ist, muss unbedingt die Abhängigkeit zu der LEDA-Bibliothek (Abschn. 2.5.3) aufgehoben werden. Diese Bibliothek führt bereits jetzt schon zu zahlreichen Einschränkungen in der Wahl der Compiler und der benutzten Bibliotheken. Mit der Ablösung von AGD (Abschn. 2.5.2) durch OGDF (Abschn. 3.3) wird es auch möglich sein, alle Abhängigkeiten zu LEDA innerhalb von *Cupe* zu beseitigen. Im Vorfeld dieser Arbeiten kann *Cupe* aber bereits jetzt schon zu der aktuellen Qt-Bibliothek (Version 4) portiert werden. Bisher wird für *Cupe* die Qt3-Bibliothek benutzt (Abschn. 2.5.4), die aber bereits jetzt schon nicht mehr von allen Linux-Distributionen unterstützt

wird. Leider sind mit dem Versionssprung von Qt auch zahlreiche Änderungen eingeführt worden, die umfangreiche Anpassungen am Source-Code von *Cupe* nach sich ziehen werden. Es ist realistisch, einen Zeitraum von 6-8 Wochen für diese Arbeiten einzuplanen.
Auch für das automatische Zeichnen von metabolischen Netzwerken gibt es bereits jetzt schon Vorschläge für weitere Entwicklungen. So existieren zum Beispiel schon Arbeiten für eine optimierte Darstellung von Nebenmetaboliten und die Zeichnung von Kreisen in metabolischen Reaktionsnetzwerken (Thelen, 2005; Menze, 2004), die noch nicht in *Cupe* integriert worden sind. Im Zuge einer Diplomarbeit innerhalb der Bioinformatik könnte auch versucht werden, den von Becker & Rojas (2001) vorgeschlagenen Algorithmus zum Zeichnen von Stoffwechselnetzwerken zu implementieren. Ebenfalls wäre es schön, für die automatische, hierarchische Zeichnung eines Graphen eine Quelle und eine Senke anzugeben, um so den Pfad von einem Substrat zu einem Produkt von oben nach unten darstellen zu können. Diese Technik ist bereits für BioPath implementiert worden (Brandenburg *et al.*, 2001) und kann möglicherweise im Rahmen eines Praktikums in der Informatik durchgeführt werden. Wenn es möglich wäre, solche *gerichtete* hierarchische Darstellungen zu erzeugen, wäre es sicher auch möglich, zwei ähnliche Pfade direkt miteinander zu vergleichen und Ähnlichkeiten auf die gleiche Weise dazustellen.
Eine weitere Idee zur Weiterentwicklung von *Cupe* betrifft die Darstellung von Netzwerken in Clustern (Abschn. 3.2.1). Bisher ist es so, dass sich der Graph in solch einem „Superknoten“ nicht verändert, wenn die Zeichnung des Graphen, der den Cluster umgibt, verändert wird. Das führt leider dazu, dass es zu ungewollten Kantenüberkreuzungen innerhalb der Clusterzeichenfläche kommt. Wenn man die Zahl der ein- und ausgehenden Kanten in diese Cluster beschränkt, dann müsste es möglich sein, den Graphen, der in einem Cluster dargestellt wird, durch einfache Spiegelung und Drehung um eine Achse an das neue Layout des umgebenen Graphen anzupassen und Kantenüberkreuzungen zu vermeiden. Da Cluster rekursiv sind, dass heißt, ihrerseits selber Cluster enthalten können und Cluster über zwei Stufen kollabiert und expandiert werden können, kann man auf diese Art z.B. einen sehr großen Graphen auf wenige Knoten und Kanten „kondensieren“ und bei Bedarf wieder expandieren. Diese Arbeit könnte vermutlich im Rahmen der Bioinformatik während einer Diplomarbeit durchgeführt werden.
Schlußendlich gibt es noch zwei kleinere Vorschläge, die vom Umfang her im Rahmen eines Praktikums durchführbar sein müssten. Erstens wäre es sicher sinnvoll, wenn man auf die Analysealgorithmen des *Pathway Hunter Tools* (Rahman S. A., 2004) auch von *Cupe* aus zugreifen könnte. Die notwendigen Schnittstellen hierzu existieren bereits und könnten über ein Plugin für *Cupe* genutzt werden. Zweitens sollte es auch möglich sein, Konzentrationsprofile oder andere Daten dieser Art, ähnlich wie im Programm VANTED (Junker *et al.*, 2006) demonstriert, auch in *Cupe* darzustellen. Hierzu müsste lediglich ein entsprechendes Plugin angeboten werden und die Information in die Zeichnung eines Knoten gerendert werden, wobei das Zeichnen beliebiger Strukturen in die Darstellung eines Knoten von der Qt-Bibliothek sehr komfortabel unterstützt wird. Von solch einer Darstellung könnten auch die bereits

entwickelten Plugins *Rationindikator* (Thielen, B., 2005) und *MPN* (Hartman, K., 2005) profitieren.

Literaturverzeichnis

Alberts, D., Gutwenger, C., Mutzel, P. & Näher, S. (1997). AGD–Library: A Library of Algorithms for Graph Drawing. In *Proceedings of the Workshop on Algorithm Engineering (WAE '97)* (Italiano, G.F. & Orlando, S, eds). Venice, Italy, pp. 112–123. `http://www.dsi.unive.it/~wae97/proceedings/`.

Alexandrescu, A. (2001). *Modern C++ Design. Applied Generic and Design Patterns.* Addison Wesley.

Algorithmic Solutions (2001). *The LEDA User Manual — Version 4.2.1.* `http://www.mpi-sb.mpg.de/LEDA/MANUAL/MANUAL.html`.

AT&T Research (2003). Graphviz - Graph Visualization Software.`http://www.graphviz.org/`

Bairoch, A (1999). The ENZYME data bank. *Nucleic Acids Res.*, **27**, 310–311, `www.expasy.ch/enzyme/`.

Bairoch, A. & Apweiler, R. (1998). The Swiss-Prot protein sequence data bank and its supplement TrEMBL. *Nucleic Acids Res.*, **26**, 38–42, `http://www.ebi.ac.uk/swissprot/`.

Becker, Moritz Y. & Rojas, Isabel (2001). A graph layout algorithm for drawing metabolic pathways. *Bioinformatics*, **17**, 461–467, .

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N. & Bourne, P. E. (2000). The Protein Data Bank. *Nucleic Acids Res.*, **28**, 235–242, `http://www.rcsb.org/pdb/`.

Brandenburg, F. J., Forster, M., Pick, A., Raitner, M. & Schreiber, F. (2001). BioPath - Visualization of Biochemical Pathways. In *Proceedings of the German Conference on Bioinformatics* (*In silico* Biology, ed.). Bioinformation Systems e.V.-Team. `http://www.bioinfo.de/isb/gcb01/talks/brandenburg/index.html`.

Breymann, U. (2002). *Komponenten entwerfen mit der C++ STL.* Addison-Wesley, Bonn. `http://www.informatik.hs-bremen.de/~brey/stlb.html`.

Consortium, The Gene Ontology (2000). Gene Ontology: tool for the unification of biology. *Nature*, **25**, 25–29.

Demir, E., Babur, O., Dogrusoz, U., Gursoy, A., Nisanci, G., Getin-Ataly, R. & Ozturk, M. (2002). PATIKA: an integrated visual environment for collaborative construction and analysis of cellular pathways. *Bioinformatics*, **18**, 996–1003, `http://www.patika.org/`.

Eades, P (1984). A heuristic for graph drawing. *Congressus Numerantium*, **42**, 391–395.

Ellis, Lynda B. M., Speedie, stuart M. & McLeish, R. (1998). Representing metabolic pathway information: an opject-oriented approach. *Bioinformatics*, **14**, 803–806, `http://umbbd.ahc.umn.edu/`.

Fröhlich, M. (1998). *Inkrementelles Graphlayout im Visualisierungssystem daVinci.* Number 6 in BISS Monographs. Shaker Verlag; Postfach 1290, 52013 Aachen; `http://www.shaker.de`.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1996). *Entwurfsmuster.* Addison-Wesley-Longman, Bonn.

Goesmann, A., Haubrock, M., Meyer, F., Kalinowski, J. & Giegerich, R. (2002). PathFinder: reconstruction and dynamic visualization of metabolic pathways. *Bioinformatics*, **18**, 124–129, `http://bibiserv.techfak.uni-bielefeld.de/pathfinder/`.

Hartman, K. (2005). Visualisation of simulation results of Metabolic Petri Nets, `http://www.cubic.uni-koeln.de/`

Himsolt, M. (1996). GML: Graph Modelling Language. `http://www.infosun.fmi.uni-passau.de/Graphlet/GML/index.html`

Hofmann, K., Bucher, P., Falquet, L. & Bairoch, A. (1999). The PROSITE database, its status in 1999. *Nucleic Acids Res.*, **27**, 215–219.

Horowitz, E. & Sahni, S. (1981). *Algorithmen. Entwurf und Analyse.* Springer-Verlag, Berlin.

Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. & Kitano, H. (2002). The ERATO Systems Biology Workbench: enabling interaction and exchange between software tools for computational biology. In *Proc. Pac Symp Biocomput.* pp. 450–61.

Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Ktano, H., Arkin, A. B., Bornstein, B. J., Bry, D., Cornisch-Bowden, A., Cuellar, A. A., Hedley, W. J., Hodgman, T. C., J.-H., Hofmey., Hunter, B. J., Juty, N. S., Kasberger, J. L., Kremling, A., U., Kummer, Le Novére, N., Loew, L. M., Lucio, D., Mendes, P., Minch, E., Mjolsness, E. D., Nakayama, Y., Neson, M. R., Nielson, P. F., Sakurada, T., Schaff, J. C., Shapiro, B. E., Shimizu, T. S., Spence, H. D., Stelling, J., Takahashi, K., Tomita, M., Wagner, J. & Wang, J. (2002). The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, **19**, 524–531, `http://www.sbml.org/`.

Hucka, M., Sauro, H, Finney, A., Bolouri, H., Doyle, J. & Kitano, H. (2001). Foundations of Systems Biology. In *First International Symposium on Computational Cell Biology, Lenox, Massachussetts, USA* (Kitano, Hiroaki, ed.). MIT Press, Cambridge.

Junker, H. B., Klukas, C. & F., Schreiber (2006). VANDTED: A system for advanced data analysis and visualization in the context of biological networks. *BMC Bioinformatics*, **7**, 109–121.

Kanehisa, M. & Goto, S. (1999). KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Res.*, **28**, 27–30, `http://www.genome.ad.jp/kegg/`.

Karlsson, B. (2005). *Beyond the C++ Standard Library. An Introduction to Boost.* Addison Wesley. `http://www.boost.org`.

Karp, P. D., Ouzounis, C. & Paley, S. (1996). Hinccyc: a Knowledge base of the complete genome and metabolic pathways of *H. influenzae.* In *Proceedings of the ISMB'96 Conference.* AAAI Press, Menlo Park, pp. 116–129.

Karp, Peter D., Paley, S. & Romero, P. (2002). The Pathway Tools software. *Bioinformatics*, **18**, 225–232, `http://bioinformatics.ai.sri.com/ptools/`.

Karp, P. D., Ridley, M., Paley, S. M., Pellegrine-Toole, A. & Krummenacher, M. (1999). EcoCyc: encyclopedia of *Escherichia coli* genes and metabolism. *Nucleic Acids Res.*, **27**, 55–58, `http://ecocyc.org`.

Küffner, R., R., Zimmer & T., Lengauer (2000). Pathway analysis in metabolic databases via differntial metabolic display (DMD). *Bioinformatics*, **16**, 825–836, `http://cartan.gmd.de/ToPLign.html`.

Lehninger, Nelson & Cox (1994). *Prinzipien der Biochemie.* Spektrum Akademischer, Heidelberg.

Mehlhorn, K. & Näher, S. (1989). LEDA: A libary of efficient data types and algorithms. In *Proceedings of the 14th International Symposium on Mathematical Foundations of Computer Science (MFCS'89), Lecture Notes in Computer Science* (Kreczar, Antonii & Mirkowska, Grazyna, eds), volume 379. Springer, pp. 88–106.

Menze, Annette (2004). Darstellung von Nebenmetaboliten in automatisch erzeugten Zeichnungen metabolischer Netzwerke. Diplomarbeit, Universität zu Köln.

Meyers, Scott (1997). *Mehr effektiv C++ programmieren.* Adison-Wesley-Longman, Bonn.

Meyers, Scott (2005). *Effektiv C++ programmieren.* Adison-Wesley, München.

Michal, G. (ed.) (1999). *Biochemical Pathways, (Biochemie-Atlas).* Spektrum Akademischer, Heidelberg. `http://www.expasy.org/tools/pathways/`.

Mutzel, P., Jünger, M. & Leipert, S. (ed.) (2001). *Graph Drawing, 9th International Symposium, GD 2001 Vienna, Austria, September 23-26, 2001. Revised Papers.* Springer, Heidelberg.

Näher, S. & Mehlhorn, K. (1990). LEDA: A libary of efficient data types and algorithms. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90), Lecture Notes in Computer Science* (Paterson, Michael S., ed.), volume 443. Springer, pp. 1–5. `http://www.algorithmic-solutions.com/`.

Näher, S. & Mehlhorn, K. (1990). LEDA: A libary of efficient data types and algorithms. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90), Lecture Notes in Computer Science* (Paterson, Michael S., ed.), volume 443. Springer, pp. 1–5. `http://www.algorithmic-solutions.com/`.

NC-IUBMB (1992). *Enzyme Nomenclature.* Academic Press, San Diego, California. `http://www.chem.qmul.ac.uk/iubmb/enzyme`.

OpenGL.org, ed. (2005). *The Official Reference Document to OpenGL.* Addison Wesley Professional, www.opengl.org. http://www.opengl.org.

Ottmann, T. & Widmayer, P. (1996). *Algorithmen und Datenstrukturen.* Spektrum Akademischer Verlag, Heidelberg.

Overbeek, R., larsen, N., Gorden, D. P., D'Souza, M., Selkov Jr, E., Kyrpides, N., Fonstein, M., Maltsev, N. & Selkov, E. (2000). WIT: integrated system for high-throughput genome sequence analysis and metabolic reconstruction. *Nucleic Acids Res.*, **28**, 123–125, http://wit.mcs.anl.gov/WIT2/CGI/.

Overbeek, R., Larsen, N., Pusch, G. D., D'Souza, Selkov Jr, E., Kyrpides, N., Fonstein, M., Maltsev, N. & Selkov, E. (1999). WIT: intergrated system for hight-throughput genome sequence analysis and metabolic reconstruction. *Nucleic Acids Res.*, **28**, 123–125, http://wit.mcs.anl.gov/WIT2/.

Paley, Suzanne M. & Karp, Peter D. (2001). Evaluation of cumputational metabolic-pahtway predictions for Helicobacter pylori. *Bioinformatics*, **18**, 715–724, http://ecocyc.org:1555/HPY/organism-summary?object=HPY.

Passau, University (2006). Gravisto is an editor for graphs and a toolkit for implementing graph visualization algorithms. http://gravisto.fmi.uni-passau.de/

Petri, Carl Adam (1962). Kommunikation mit Automaten. *Bonn: Institut für Instrumentelle Mathematik*, **Nr. 3**, http://www.daimi.au.dk/PetriNets/faq/.

Pfeiffer, T., Sánches-Valdenebro, I., Nuño, J. C., Montero, F. & Schuster, S. (1999). METATOOL: for studying metbolic networks. *Bioinformatics*, **15**, 251–257, http://www.bioinf.mdc-berlin.de/projects/metabolic/metatool/.

PubMed (1965 – 2003). PubMed, a service of the National Library of Medicine, provides access to over 12 million MEDLINE citations back to the mid-1960's and additional life science journals. http://www.ncbi.nlm.nih.gov/entrez/query.fcgi

Rahman S. A. (2004). Pathway Hunter Tool: Comparative Study of Metabolic Networks - An In Silico Systems Biology. http://www.pht.uni-koeln.de

Sander, G. (1995). Graph layout through the vcg tool. In *DIMACS International Workschop GD'94. Proceedings, Lecture Notes in Computer Science 894* (In Tamassia, I.G.T.R., ed.). Springer, Berlin, pp. 194–205.

Schomburg, I., Hofmann, O., Bänsch, C., Chang, A. & Schomburg, D. (2000). Enzyme data and metabolic information: Brenda, a resource for research in biology, biochemistry, and medicine. *Gene Funct. Dis.*, **3-4**, 109–18, http://www.brenda.uni-koeln.de/.

Selkov, E., Basmanova, S., Gaasterland, T., Goryanin, I., Gretchkni, Y., Meltsev, N., Nenashev, V., Overbeek, R., Panyushkina, E., L.Pronevitch, Selkov, E. & Yunis, I. (1996). The metabolic pathway collection from EMP: The enzymes and metabolic pathways database. *Nucleic Acids Res.*, **24**, 26–28, http://wit.mcs.anl.gov/EMP/.

Selkov, E., Galimova, M., Goryanin, I., Gretchkin, Y., Ivanova, N., Komarow, Y., Maltsev, N., Mikhailova, N., Nenashev, V, Overbeek, R., Panyuskina, E., Pronevitch, L. & Selkov, E. Jr. (1997). The metabolic Pathway collection: an update. *Nucleic Acids Res.*, **25**, 37–38, http://wit.mcs.anl.gov/WIT2/.

Selkov, E. J., Grechkin, Y., Mikhailov, N. & Selkov, E. (1998). MPW: the metabolics pathway database. *Nucleic Acids Res.*, **26**, 43–45, `http://wit.mcs.anl.gov/MPW/`.

Shannon, B., Markiel, A., Ozier, O., Baliga, N. S., Wang, J. T., Ramage, D., Amin, N., Schwikowski, B. & Ideker, T. (2003). Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks. *Genome Research*, **13**, 2498–2504.

Stösser, G., Tuli, M.A., Lopez, R. & Sterk, P. (1999). The EMBL nucleotide sequence database. *Nucleic Acids Res.*, **27**, 18–24, `http://www.ebi.ac.uk/embl/`.

Stroustrup, B. (2000). *Die C++ Programmiersprache. Deutsche Ausgabe der Special Edition. 4. Auflage*. Addison-Wesley Verlag, München.
`http://www.research.att.com/~bs/`.

Sutter, H (2000). *Exceptional C++*. Addison-Wesley, Germany.

Thelen, C. (2005). Darstellung von Kreisen in orthogonalen Zeichnungen von Graphen. Diplomarbeit, Universität zu Köln.

Thielen, B. (2005). CUPE Metabolic Ratio Indicator Plugin,
`http://www.uni-koeln.de/math-nat-fak/biochemie/ds/dsak_d.htm`

Trolltech (2003). QT: A multiplatform, C++ application development framework.
`ftp://ftp.trolltech.com/qt/pdf/3.1/qt-whitepaper-a4-web.pdf`

Trost, E., Hackl, H., Maurer, M. & Trajanoski, Z. (2002). Java editor for biological pathways. *Bioinformatics*, **19**, 786–787,
`http://genome.tugraz.at/Theses/Trost2002.pdf`.

Vandervoorde, D. & M., Josuttis N. (2003). *C++ Templates. The Complete Guide*. Addison-Wesley, Boston.

Šīrava, M, Schäfer, T., M., Eiglserger, M., Kaufmann, Kohlbacher, O., Bornberg-Bauer, E. & P., Lenhof H. (2002). BioMiner—modeling, analyzing, and visualizing biochemical pathways and networks. *Bioinformatics*, **18**, 219–230,
`http://www.zbi.uni-saarland.de/chair/projects/BioMiner/index.shtml`.

Wong, L. (2001). A protein interaction extraction system.
`(citeseer.nj.nec.com/wong01protein.html)`. In *Pacific Symposium on Biocomputing 6*.

A Erklärung gemäß § 3 Abs. 10 der Promotionsordnung

Ich versichere, dass ich die von mir vorgelegte Dissertation selbständig angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie – abgesehen von unten angegebenen Teilpublikationen – noch nicht veröffentlicht worden ist sowie, dass ich eine solche Veröffentlichung vor Abschluss des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen der Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Herrn Prof. Dr. Dietmar Schomburg betreut worden.

Köln, den 12. Dezember 2006.

B

Lebenslauf

Daten zur Person

Name	Ralph Oliver Schunk
Straße	Rethelstrasse 156
Ort	40237 Düsseldorf
Telefon	+49 (0)211 5988080
Geboren/Ort	07.04.1972 in Meerbusch-Lank (Land Nordrhein–Westfalen)
Familienstand	ledig

Schulbildung

08/78–06/82	Sankt-Paulus-Grundschule in Düsseldorf
08/82–06/91	Heinrich-Heine-Gesamtschule
Schulabschluß	Allgemeine Hochschulreife

Wehrersatzdienst

11/91–02/93	Mobile soziale Hilfsdienste (Jugendamt Düsseldorf)

Studium

seit 10/93	Studium der Biologie Heinrich-Heine-Universität Düsseldorf

Beurlaubung

04/94–10/94	Organisation und Durchführung der Jugendfreizeit in der Kirchengemeinde St. Paulus

Beurlaubung	
04/95–10/95	Organisation und Durchführung der Jugendfreizeit in der Kirchengemeinde St. Paulus
10/96	Diplom-Vorprüfung
Hauptstudium	
10/96–12/01	Physikalische Biologie, Parasitologie, Informatik, Mathematik für Biologen, Technische Biochemie, Botanik
Unterbrechung	
05/98–10/98	Sportunfall
10/99–04/01	Beurlaubung Vorbereitung zur Diplomprüfung
12/00	Mündliche Diplomprüfungen – Physikalische Biologie – Zoologie – Informatik
03/01–12/01	Diplomarbeit am Institut für Physikalische Biologie „Simulation kinetisch kontrollierter RNA-Strukturbildung“ Gesamtnote: sehr gut
Wissenschaftliche Tätigkeit	
02/02–05/02	Wissenschaftliche Hilfskraft Heinrich-Heine-Universität Düsseldorf Institut für Physikalische Biologie
05/02–12/02	Wissenschaftlicher Angestellter Heinrich-Heine-Universität Düsseldorf Institut für Physikalische Biologie
01/03–03/07	Wissenschaftlicher Mitarbeiter Universität zu Köln Cologne University BioInformatics Center
Schüler-/ Studentenjobs	
07/91–10/91	Schüler- und Studentenvermittlung des Arbeitsamtes
10/91–12/06	Angestellt bei B+B Parkhaus GmbH & Co. KG ca. 12 Wochenstunden
03/93–06/93	Schüler- und Studentenvermittlung des Arbeitsamtes
02/94–03/94	Studentenjob bei Henkel KGaA (Vervielfältigung)
01/01–03/01	Schüler- und Studentenvermittlung des Arbeitsamtes

04/01–12/01	Studentische Hilfskraft im Institut für Physikalische Biologie. Heinrich-Heine-Universität Düsseldorf
Semesterferien	Dienstleistungsbetriebe, Groß- und Einzelhandel, Speditionen, Lagerhaltung, ...

Sonstiges

07/93–08/93	Fahrradtour durch die Schweiz (8 Wochen)
10/94–10/97	Ordentlich gewähltes Mitglied des Fachschaftsrates Biologie
06/95–03/02	Tätigkeit als Nachhilfelehrer

Düsseldorf, 12. Dezember 2006